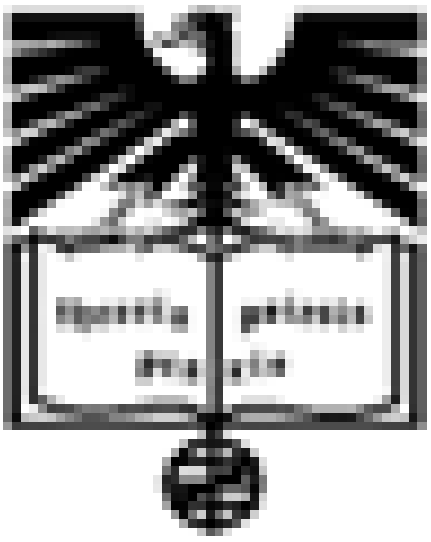
Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática
2009

João Miguel Matos Baptista Santos

Implementação em hardware de receptor para redes ópticas de transporte



João Miguel Matos Baptista Santos

Implementação em hardware de receptor para redes ópticas de transporte

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica, realizada sob a orientação científica da professora Dra. Iouliia Skliarova, Professora auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro e Victor Abreu, da empresa Withus

o júri

presidente

Professor Doutor José Carlos Esteves Duarte Pedro
professor catedrático da Universidade de Aveiro

Professor Doutor António José Duarte Araújo
professor auxiliar da Faculdade de Engenharia da Universidade do Porto

Professora Doutora Iouliia Skliarova
professora auxiliar da Universidade de Aveiro (orientadora)

agradecimentos

É com muito gosto que aproveito esta oportunidade para agradecer a todos os que me ajudaram durante este longo ano de trabalho.

Em particular gostaria de agradecer aos orientadores Iouliia Skliarova e Victor Abreu que sempre mostraram disponibilidade em ensinar e dar sugestões. Também gostaria de agradecer ao Bruno Monteiro, pela ajuda preciosa em partes do projecto.

Por último um agradecimento especial à minha família pelo apoio moral e financeiro.

palavras-chave

Comunicações ópticas, redes ópticas de transporte (OTN), recomendação ITU-T G.709, dispositivos reconfiguráveis, simulação de hardware, *Field Programmable Gate Array* (FPGA), implementação em *hardware*, códigos correctores de erros, Reed-Solomon(255,239)

resumo

Com a crescente procura de largura de banda, a que se assiste actualmente, torna-se necessário encontrar novas soluções tanto a nível da camada física como a nível da camada protocolar. A norma ITU-T G.709, também conhecida por OTN (*Optical Transport Network*), vem resolver muitos dos problemas resultantes do aumento do ritmo de transmissão de dados sobre fibra óptica. Esta norma traz grandes vantagens para os operadores, nomeadamente uma maior transparência, compatibilidade com diversos protocolos existentes, monitorização mais eficaz do tráfego e principalmente a inclusão de um poderoso código corrector de erros FEC (*Forward Error Correction*).

O objectivo deste projecto é implementar um dispositivo em *hardware* capaz de receber dados segundo a norma OTN. As tarefas a desempenhar pelo receptor são: sincronização da trama, monitorização do tráfego através da análise dos principais cabeçalhos da trama e correcção de erros usando o código FEC. O presente trabalho foca-se na descrição e simulação dos blocos necessários à implementação do sistema, na síntese dos mesmos para determinar quais os recursos necessários e na verificação dos requisitos temporais.

keywords

Optical communications, Optical Transport Networks (OTN), recommendation ITU-T G.709, reconfigurable devices, hardware simulation, Field Programmable Gate Array (FPGA), hardware implementation, error-correcting codes, Reed-Solomon(255,239)

abstract

With the increasing demand for bandwidth, it is necessary to find new solutions at the level of both physical and protocol layer. The ITU-T G.709 recommendation, also known as Optical Transport Network (OTN), allows to solve many of the problems caused by the increase of data transmission rate in optical fiber. This recommendation presents great advantages to network operators, including protocol transparency, backward compatibility with existing protocols, more efficient traffic management and, especially, the inclusion of a powerful Forward Error Correction (FEC) code.

The purpose of this project is to implement a device in hardware capable of receiving data according to the OTN recommendation. The tasks to be performed by the receiver are: frame synchronization, traffic monitoring through the analysis of the main overheads and error correction using the FEC code. This work focuses on the description and simulation of the blocks needed to implement the system, on the synthesis of these blocks to determine the resources needed and on verification of timing constraints.

Conteúdo

Lista de Tabelas

Lista de Figuras

Lista de Siglas

0xYY	YY é um valor em representação hexadecimal
3R	*Retiming, Reshaping and Reamplification*
APS	*Automatic Protection Switching*
ARQ	*Automatic Repeat Request*
ASIC	*Application-Specific Integrated Circuit*
AU-4	*Administrative Unit - level N*
AUG	*Administrative Unit Group*
BDI	*Backward Defect Indication*
BEI	*Backward Error Identifier*
BER	*Bit Error Rate*
BIAE	*Backward Incoming Alignment Error*
BIP	*Bit Interleaved Parity*
BM	*Berlekamp-Massey*
CG	Corpo de Galois
CLB	*Configurable Logic Block*
CMT	*Clocking & Management Tile*
CWDM	*Coarse Wavelength Division Multiplex*
DAPI	*Destination Access Point Identifier*
DCM	*Digital Clock Manager*
DC	*Discrepancy Computation*
DFS	*Digital Frequency Synthesizer*
DLL	*Delay-Locked Loop*
DWDM	*Dense Wavelength Division Multiplex*
DW	*Digital Wrapper*

EDFA *Erbium Doped Fiber Amplifiers*

ELU *Error Locator Update*

EXP *Experimental*

FA OH *Frame Alignment OverHead*

FAS *Frame Alignment Signal*

FEC *Forward Error Correction*

FPGA *Field Programmable Gate Array*

FTFL *Fault Type & Fault Location report channel*

GbE *Gigabit Ethernet*

GCC *General Communication Channel*

IaDI I*ntra-domain Interface*

IAE *Incoming Alignment Error*

iBM *inversionless* BM

IP *Internet Protocol*

IrDI *Inter-domain Interface*

ITU *International Telecommunication Union*

ITU-T ITU-*Telecommunication standardization sector*

JC *Justification Control*

LDPC *Low-Density Parity-Check*

LFSR *Linear Feedback Shift Register*

LSB *Least Significant Bit*

LUT *Lock Up Table*

MDC Máximo Divisor Comum

MFAS *Multi-Frame Alignment Signal*

MSB *Most Significant Bit*

MSOH *Multiplex Section OverHead*

NJO *Negative Justification Opportunity*

OADM *Optical Add-Drop Multiplexer*

OAM&P *Operations, Administration, Maintenance and Provisioning*

OC-N *Optical Carrier - level N*

OCC *Optical Channel Carrier*

OCh *Optical Channel*

OCX *Optical Crossconnector*

ODU *Optical channel Data Unit*

OEO *Optical-Electrical-Optical*

OH *OverHead*

OLT *Optical Line Terminal*

OMS *Optical Multiplex Section*

OPU *Optical channel Payload Unit*

OTN *Optical Transport Network*

OTS *Optical Transmission Section*

OTU *Optical channel Transport Unit*

PCC *Protection Communication Control channel*

PDH *Plesiochronous Digital Hierarchy*

PJO *Positive Justification Opportunity*

PLL *Phase-Locked Loop*

PM *Path Monitoring*

POH *Path OverHead*

PSI *Payload Structure Identifier*

PS *Phase Shift*

PT *Payload Type*

QoS *Quality of service*

RES *Reserved*

RSOH *Regenerator Section OverHead*

RS *Reed-Solomon*

SAPI *Source Access Point Identifier*

SDH *Synchronous Digital Hierarchy*

SERDES *SERializer/DESerializer*

SM *Section Monitoring*

SONET *Synchronous Optical Networking*

STM-N *Synchronous Transport Module - level N*

STS-N *Synchronous Transport Signal - level N*

TCM *Tandem Connection Monitoring*

TDM *Time Division Multiplexing*

TTI *Trail Trace Identifier*

VC-N *Virtual Container level-N*

VHDL VHSIC *Hardware Description Language*

VHSIC *Very High Speed Integrated Circuits*

VoIP *Voice over Internet Protocol*

WDM *Wavelength Division Multiplex*

Capítulo 1

Introdução

Com o aumento da globalização, a troca de informação tornou-se uma actividade fundamental no andamento da economia, impulsionando a formação de redes metropolitanas rápidas, flexíveis e confiáveis. Com a demanda sempre crescente por novos serviços, a capacidade de transferir informação diversificada e de diferentes níveis de complexidade na mesma infra-estrutura tornou-se o objectivo das operadoras de telecomunicações que prevêem uma acentuada procura por serviços que exigem grande largura de banda, como vídeo-conferência, educação à distância, telemedicina, voz sobre IP (VoIP), entre outros.

A crescente necessidade de largura de banda incentivou a criação da recomendação G.709, também conhecida por OTN (*Optical Transport Network*) pelo ITU-T (*International Telecommunication Union - Telecommunication standardization sector*), que define as interfaces para redes ópticas de transporte com ritmos de transmissão até $40Gb/s$. A recomendação OTN é a base das redes ópticas da próxima geração e pretende combinar as funcionalidades de supervisão dos protocolos SONET/SDH, com a expansibilidade de largura de banda das redes DWDM (*Dense Wavelength Division Multiplex*). A norma OTN é definida pelo ITU-T como um conjunto de elementos ópticos de rede, ligados por fibra óptica, capazes de fornecer funcionalidades de transporte, multiplexagem, encaminhamento, manutenção e monitorização dos canais ópticos que transportam os sinais provenientes do cliente [IT03b].

A norma OTN trouxe algumas funcionalidades novas que permitem melhorar o desempenho da rede. O factor que mais valoriza esta nova recomendação é a inclusão de um mecanismo de correcção de erros robusto, que permite uma maior separação entre regeneradores e consequentemente uma redução de custos. A norma define um código corrector de erros *Reed-Solomon* não binário, que é largamente utilizado em aplicações comerciais (CD, DVD, Blu-Ray ou DVB). Os códigos *Reed-Solomon* são especialmente eficientes a lidar com longas sequências de bits errados e por isso são adequados para comunicações ópticas. O princípio de funcionamento destes códigos de erros baseia-se na teoria dos corpos finitos ou corpos de Galois.

A motivação do presente trabalho surgiu a partir da proposta lançada pelas empresas *Withus* e *PT Inovação*, que desenvolvem equipamentos de telecomunicações. O projecto visa desenvolver um bloco de multiplexagem que fará parte do sistema MUX-OTN. Mais concretamente pretende-se desenvolver um equipamento terminal de recepção de dados segundo a norma G.709.

1.1 Objectivos

Em termos gerais o objectivo deste trabalho é implementar em FPGA, um sistema capaz de receber sinais OTN, a um ritmo de transmissão de aproximadamente $2,7Gb/s$. O sistema

deverá ser capaz de efectuar o sincronismo e de seguida proceder à análise dos campos mais importantes do cabeçalho. Por último o sistema deve corrigir possíveis erros ocorridos durante a transmissão. Mais detalhadamente, os objectivos a atingir através da elaboração desta tese são:

- Estudar a linguagem de descrição de hardware VHDL;
- Aprender a usar as ferramentas ISE e *ModelSim*;
- Estudar as características dos FPGAs *Virtex-5* da *Xilinx*;
- Analisar em detalhe a recomendação G.709 (OTN) e comparar com o protocolo SDH;
- Estudar os códigos de erros, em particular o código *Reed-Solomon(255,239)* usado na norma OTN;
- Estudar as várias soluções para o receptor OTN e implementar a solução escolhida utilizando VHDL;
- Validar a implementação escolhida recorrendo a simulações em *ModelSim*.

Um objectivo futuro será testar o funcionamento do sistema na plataforma física (FPGA), usando fluxos de dados reais.

Um terminal para redes OTN pressupõe a existência de um bloco de transmissão e outro de recepção. Tanto o bloco de transmissão como o de recepção podem ser divididos em vários blocos mais simples. A figura 1.1 ilustra a arquitectura genérica de um dispositivo terminal para redes OTN. Os blocos a implementar neste trabalho encontram-se a cinzento. A interface entre os domínios óptico e eléctrico não faz parte deste projecto, por isso assume-se que os dados de entrada do receptor já estão no domínio eléctrico. Os blocos da parte superior do diagrama fazem parte do terminal de recepção e como tal são os blocos principais deste projecto. No entanto, será também implementado o bloco de codificação FEC, pois será útil na fase de simulação do descodificador. Os blocos *Descrambler* e *Scrambler* são iguais. O bloco de acesso à carga-paga não faz parte deste projecto.

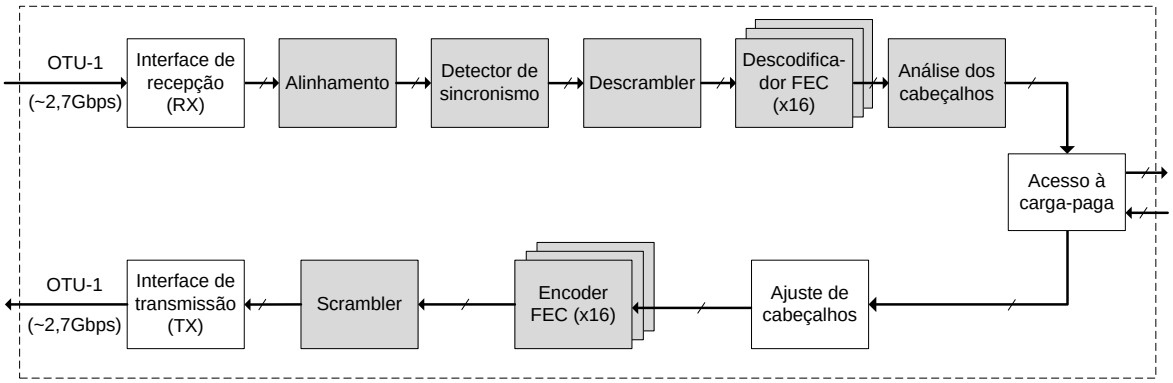


Figura 1.1: Diagrama de blocos genérico de um terminal para redes OTN

1.2 Organização da dissertação

Além deste primeiro capítulo introdutório, a presente dissertação encontra-se dividida em mais 4 capítulos. O capítulo 2 pretende dar uma perspectiva geral sobre as comunicações

ópticas actuais, bem como alguns aspectos teóricos. No capítulo 3 são descritos os blocos que formam o sistema, enquanto no capítulo 4 os blocos criados são validados, recorrendo a simulações. As conclusões são apresentadas no capítulo 5.

No capítulo 2 fala-se nos principais protocolos de comunicação sobre fibra óptica. Em mais detalhe, é abordada a norma OTN, que é o tema principal deste texto. Sobre a norma OTN são descritas as suas principais características e vantagens relativamente a outros protocolos. É ainda apresentado o seu funcionamento geral, analisando a hierarquia, a trama e os ritmos de transmissão. Uma grande parte deste capítulo é dedicada ao estudo dos códigos de erros, tendo em vista adquirir os conhecimentos teóricos necessários para implementar o bloco de correcção de erros. Por último, é feita a comparação com outros trabalhos da mesma área.

O capítulo 3 é dedicado à implementação do sistema. Neste capítulo começa-se por indicar as principais características dos FPGAs *Virtex-5* da *Xilinx* e referem-se alguns aspectos a ter em conta no projecto de sistemas digitais. Relativamente à implementação propriamente dita, é descrito cada um dos blocos que compõem o sistema através de diagramas de blocos ou diagramas de fluxo de sinal. São também apresentados os resultados da síntese de cada bloco e do sistema integrado, indicando os recursos necessários e a frequência máxima de operação.

No capítulo 4 é efectuada a validação de cada bloco individualmente e também do sistema integrado. A validação é feita principalmente através de simulações no programa *ModelSim*. Os resultados das simulações são apresentados em forma de diagramas temporais, que são analisados em detalhe. Em algumas situações mais complexas os resultados são comparados com resultados obtidos em *MATLAB*, de modo a confirmar valores.

Finalmente no capítulo 5 são apresentadas as principais conclusões deste trabalho. É apresentado um resumo de todo o trabalho realizado e são feitas algumas propostas de trabalho futuro relacionado com este projecto.

Capítulo 2

Estado da arte

Desde os tempos antigos que uma das principais necessidades do homem é comunicar. Nos primórdios, os sistemas de comunicação eram muito rudimentares e usados apenas para transmitir pequenas mensagens ou avisos. Eram sistemas com taxas de transmissão muito baixas e baseados somente em técnicas acústicas ou ópticas, como por exemplo cornetas e sinais de fumo. Ao longo dos tempos o Homem tem dedicado grande parte do seu tempo a desenvolver novas técnicas de comunicação. Várias tecnologias foram surgindo, mas sempre com os mesmos objectivos: aumentar o alcance, a taxa de transmissão e a fiabilidade da comunicação.

2.1 Evolução dos sistemas de comunicação

Um dos maiores avanços tecnológicos na área das comunicações ocorreu na década de 1830 quando Samuel F. B. Morse inventou o telégrafo, dando início à era das comunicações eléctricas. Nos anos seguintes assistiu-se a uma grande evolução e não demorou muito até surgir a primeira comunicação sem fios, feita no final do século XIX. Ao longo do século XX o espectro electromagnético foi ficando cada vez mais ocupado, isto porque a taxa de transmissão está directamente relacionada com a gama de frequências utilizada. Aumentar a frequência da portadora teoricamente aumenta a banda disponível e consequentemente a quantidade de informação que se pode transmitir [Kei00]. Assim, a tendência dos sistemas de comunicação sempre foi utilizar frequências cada vez mais elevadas, que possam oferecer uma maior largura de banda.

Nos últimos anos tem-se verificado que os sistemas de comunicação baseados em sinais eléctricos estão a atingir o limite ou a sua evolução tem sido pequena, tornando-se necessário desenvolver outros tipos de sistemas. Foi então que surgiram as primeiras comunicações por fibra óptica. Os sistemas de comunicação ópticos além de permitirem grandes taxas de transmissão têm uma baixa atenuação, o que os torna ideais para comunicações a longas distâncias, nomeadamente ligações intercontinentais quer sejam terrestres ou submarinas. A figura 2.1 mostra a ocupação do espectro electromagnético desde as frequências audíveis até à banda ultra-violeta. Em comunicações sobre fibra óptica, as portadoras operam sobretudo na faixa do infravermelho (entre os 800 e os 1600nm), ou seja, em frequências da ordem das centenas dos *Terahertz*. Este facto permite taxas de transmissão muito elevadas (na ordem das centenas de Gb/s) e com grande margem de evolução.

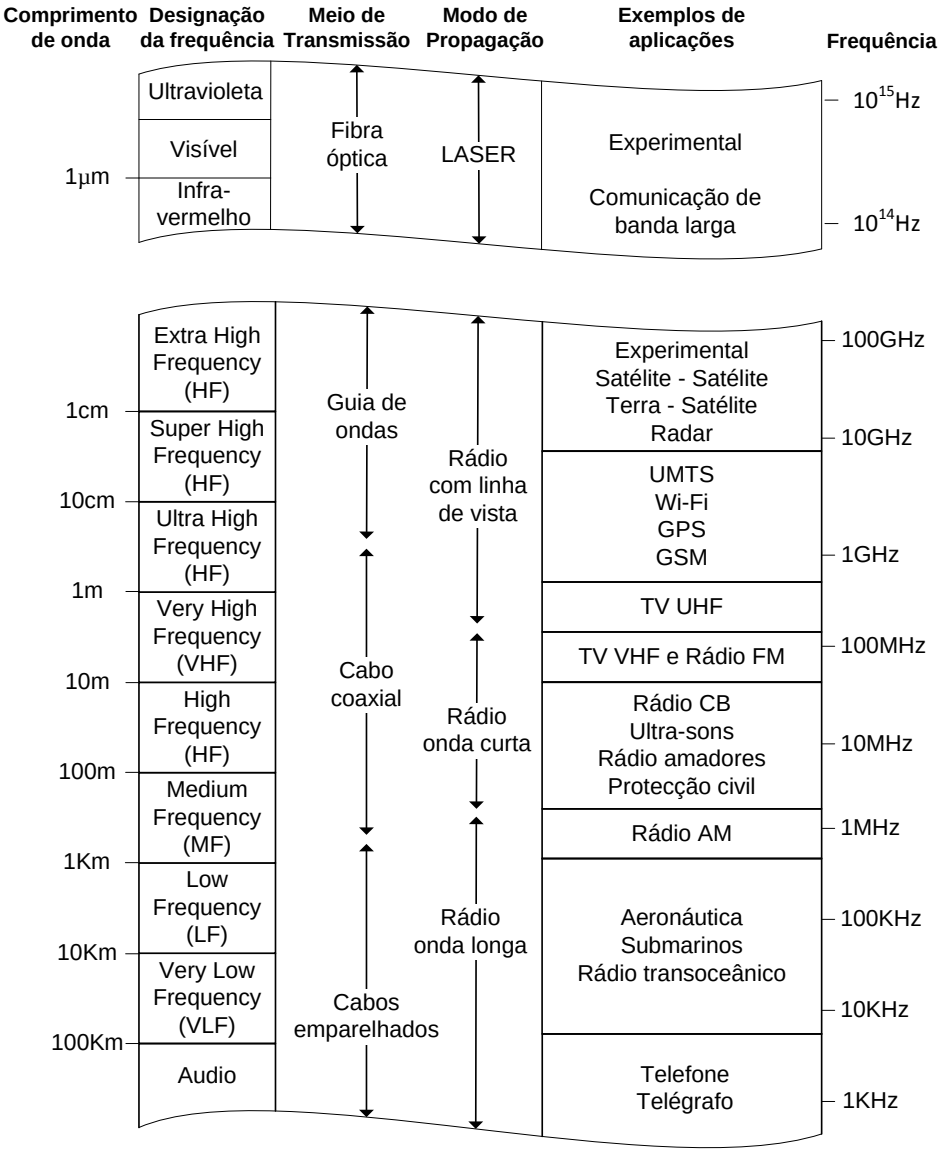


Figura 2.1: Ocupação do espectro electromagnético [CCR02]

2.1.1 Comunicações ópticas actuais

A utilização de fibra óptica para transmissão de dados veio resolver alguns problemas criados pela crescente procura de largura de banda por parte das empresas de telecomunicações. Além de oferecer uma taxa de transmissão muito superior à dos cabos de cobre ou coaxiais, é menos susceptível a interferências electromagnéticas e é mais leve. Devido a estas características, têm-se desenvolvido sistemas de comunicação ópticos para várias aplicações pontuais, como por exemplo cablagens de aviões e carros, no entanto a principal aplicação da fibra óptica continua a ser os sistemas de comunicação de elevada taxa de transmissão a longas distâncias.

Ao contrário dos sistemas eléctricos, a informação em sistemas ópticos não é transportada por modulação de frequência, mas sim por variação da intensidade luminosa. Tal como nos sistemas eléctricos o meio de comunicação pode ser o espaço livre ou um guia de ondas [Kei00], no entanto comunicações ópticas em espaço livre são muito limitadas, pois exigem que exista uma linha de vista entre o transmissor e o receptor. O outro meio de transmissão (fibra óptica) é sem dúvida o mais utilizado e explorado, existindo vários protocolos de comunicação bem

conhecidos e utilizados por todo o mundo.

A instalação de cabos submarinos permitiu um grande aumento do volume de comunicações intercontinentais, mas a certa altura houve necessidade de aumentar ainda mais a capacidade. A instalação de novos cabos de fibra óptica é dispendiosa e após o esgotamento da capacidade, tentou-se rentabilizar a infra-estrutura existente. Uma solução é aumentar o ritmo de transmissão, mas este aumento está limitado pelos dispositivos electrónicos existentes. Outra hipótese é transmitir vários fluxos de informação (canais) pela mesma fibra. Esta tecnologia designa-se por *Wavelength Division Multiplex* (WDM) ou *Coarse* WDM (CWDM) e assemelha-se à multiplexagem em frequência dos sistemas de rádio frequência. A secção de telecomunicações do *International Telecommunications Union* (ITU-T), define na norma *G.694.2* [IT03a] um espaçamento de $20nm$ entre cada canal, para comprimentos de onda centrais entre os $1271nm$ e os $1611nm$, podendo a gama ser expandida.

Mais recentemente a tecnologia evoluiu para *Dense* WDM (DWDM), que tem espaçamentos entre canais muito inferiores, o que possibilita um maior número de canais por fibra. A norma *G.694.1* [IT02] do ITU-T define espaçamentos de 12,5GHz, 25GHz, 50GHz e 100GHz para uma frequência de referência de 1937,1THz (1552,524nm). Esta gama de operação permite o uso de amplificadores ópticos dopados com Érbio (*Erbium Doped Fiber Amplifiers* - EDFA) que vêm substituir os dispendiosos regeneradores optoelectrónicos (*Optical-Electrical-Optical* (OEO) *regenerators*). Com a tecnologia DWDM, surgiu um novo conjunto de dispositivos como *Optical Line Terminals* (OLTs), *Optical Add-Drop Multiplexers* (OADM) e *Optical Crossconnectors* (OCXs), que permitem efectuar multiplexagem e comutação de canais no domínio óptico, sem necessidade de conversão para o domínio eléctrico [Pir04].

Cada um dos canais ópticos transmitidos na fibra é multiplexado no tempo (TDM - *Time Division Multiplexing*), o que permite a transmissão de sinais de baixo ritmo em canais de alto débito. A multiplexagem/desmultiplexagem é efectuada no domínio eléctrico segundo um protocolo, como por exemplo SDH (*Synchronous Digital Hierarchy*). A Hierarquia Digital Síncrona (SDH) é actualmente a norma adoptada pelo ITU-T e usada na Europa e grande parte do mundo. Teve origem no protocolo SONET (*Synchronous Optical Networking*) que, apesar de ser mais antigo, é actualmente considerado a excepção e usado apenas na América do Norte. Os protocolos SONET e SDH são muito semelhantes, variando sobretudo a nomenclatura. Neste texto será dada mais importância à norma SDH, sendo utilizada a terminologia definida pela recomendação G.707 [IT07] do ITU-T.

Os protocolos SONET e SDH vieram substituir a Hierarquia Digital Plesiócrona (PDH), também designada de assíncrona, onde o processo de multiplexagem/desmultiplexagem é bastante complexo, sobretudo para ritmos de transmissão elevados. Os protocolos síncronos, além de permitirem a utilização de hardware mais simples, trouxeram também novas funcionalidades de administração, manutenção e supervisão da informação transferida, através da inclusão de um poderoso cabeçalho entrelaçado com a carga paga.

Mais recentemente surgiu um novo protocolo para redes ópticas de transporte, o OTN *Optical Transport Network*, que pretende combinar as vantagens dos protocolos SONET/SDH com a expansibilidade de largura de banda do DWDM. Resumidamente o OTN aplica as funcionalidades de administração, manutenção e supervisão (*OAM & P - Operations, Administration, Maintenance and Provisioning*) das normas SONET/SDH às redes ópticas DWDM. Este novo protocolo é definido pela recomendação ITU-T G.709 [IT03b] e é por vezes designada de invólucro digital (DW - *Digital Wrapper*). As principais características e vantagens relativamente a outros protocolos são:

- É um protocolo transparente;
- Compatibilidade com os protocolos existentes;

- Utiliza uma poderosa técnica de correcção de erros FEC (*Forward Error Correction*);
- Reduz a necessidade de regeneradores 3R (*Retiming, Reshaping and Reamplification*).

O último ponto é particularmente importante pois minimiza a complexidade da rede, o que leva a uma redução de custos. Esta redução de equipamento é conseguida utilizando o FEC que, ao permitir a correcção de erros, introduz um ganho extra ao sistema, permitindo que os regeneradores possam estar mais espaçados [Sch06].

Nas secções seguintes serão abordados os principais protocolos usados em comunicações ópticas, desde o antigo PDH até ao recente OTN, tendo em vista compreender a razão do aparecimento deste novo protocolo.

2.1.2 Hierarquias Digitais Plesiócronas (PDH)

Para perceber os factores que levaram ao aparecimento do SDH/SONET, convém conhecer os mecanismos de multiplexagem existentes até então. Na década de 1960, a tecnologia usada era baseada na Hierarquia Digital Plesiócrona [1] (PDH), também chamada de assíncrona. Nessa altura, as técnicas de multiplexagem digital eram utilizadas essencialmente para tráfego de voz. Por essa razão, cada canal individual tinha uma taxa de transmissão de $64Kb/s$, resultante da digitalização de um circuito de voz analógico de largura de banda $4KHz$. O canal analógico é amostrado ao dobro da frequência máxima (teorema da amostragem de Shannon-Nyquist), ou seja, 8000 vezes por segundo. A digitalização é feita com 8 bits por amostra, perfazendo um canal de $64Kb/s$. A duração da trama é 1/8000, ou seja, $125\mu s$.

Nível de Multiplexagem	América do Norte (Mb/s)	Europa (Mb/s)	Japão (Mb/s)
0	0,064	0,064	0,064
1	1,544	2,048	1,544
2	6,312	8,448	6,312
3	44,736	34,368	32,064
4	139,264	139,264	97,728

Tabela 2.1: Taxas de transmissão da hierarquia PDH em várias partes do mundo [RS02]

Canais com maior taxa de transmissão são obtidos agregando vários canais fundamentais, tendo taxas de transmissão múltiplas de $64Kb/s$ [RS02]. Na América do Norte o canal fundamental é designado por DS0, enquanto na Europa é chamado de E0. Os múltiplos do canal fundamental são designados por DS1, DS2, E1, E2 e assim por diante. A tabela 2.1 mostra os ritmos de transmissão mais baixos da hierarquia PDH para vários níveis de multiplexagem, em diferentes zonas do mundo.

A tecnologia PDH funciona relativamente bem para ritmos de transmissão baixos, no entanto evidencia graves problemas para ritmos de transmissão mais elevados, tendo acabado por ser progressivamente abandonada. Para tentar eliminar os principais problemas do PDH, surgiram nos finais dos anos 80 os já mencionados SONET e SDH. De notar também que os ritmos de transmissão do PDH, como se pode ver na tabela 2.1, não são múltiplos exactos do canal fundamental, isto porque esta hierarquia faz uso extensivo de bits de aposição para compensar as diferenças de ritmo dos diversos tributários. Por isso se diz que é uma hierarquia assíncrona. Esta característica obriga a processar sucessivamente os sinais de ritmo mais

[1] O termo plesiócrono é derivado do grego *plesio*, que significa "quase" e *chronos*, que significa "tempo". Um sinónimo de plesiócrono será "quase síncrono".

elevado, descendo na hierarquia até se obter o sinal pretendido. A figura 2.2 ilustra o problema de extrair um sinal de baixo ritmo (E1) a partir de um sinal de alto ritmo (E4) [Roc07b].

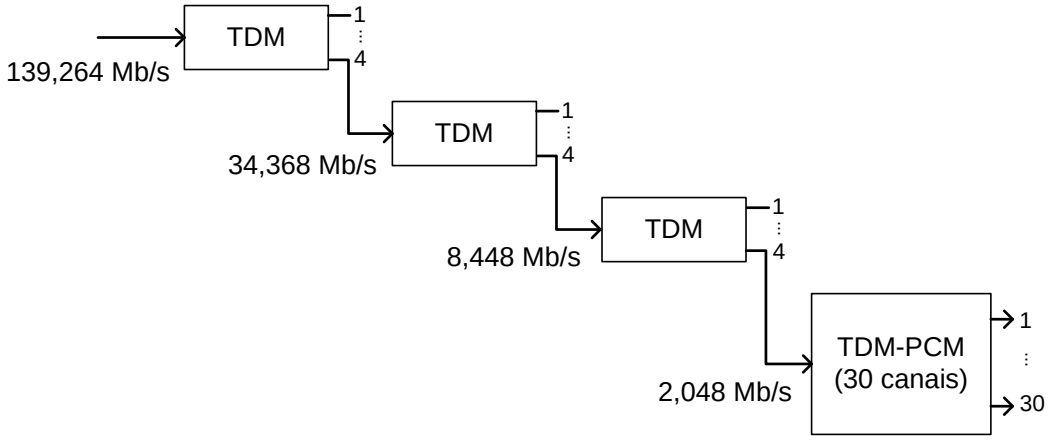


Figura 2.2: Processo de desmultiplexagem de sinais E1 a partir de sinais E4 (PDH)

Este processamento intensivo torna os equipamentos muito complexos para altas taxas de transmissão e seria mesmo impraticável desmultiplexar sinais com ritmos de transmissão equivalentes aos actuais $10Gb/s$ e $40Gb/s$ do SDH.

2.2 Hierarquias Digitais Síncronas (SDH/SONET)

Para resolver os problemas de implementação da Hierarquia Digital Plesiócrona, o SONET e o SDH utilizam outra técnica de multiplexagem, que não é baseada em bits de aposição, mas sim em ponteiros e carga paga (*payload*). Este método permite equipamento muito mais simples para aceder a canais de baixo ritmo, permitindo utilizar taxas de transmissão mais elevadas. O ponteiro de carga paga é um número transportado no cabeçalho (OverHead - OH), que indica onde começa o *Virtual Container* (VC) dentro da trama. Assim o VC não necessita de ter uma posição fixa dentro da estrutura.

2.2.1 Características gerais

Tal como o PDH, também o SONET e SDH possuem diversos níveis de multiplexagem, sendo que o nível fundamental do SONET tem uma taxa de transmissão de $51{,}84Mb/s$ e é designado no domínio eléctrico por STS-1 (*Synchronous Transport Signal level-1*) e OC-1 (*Optical Carrier level-1*) se convertido para o domínio óptico. Por sua vez o sinal fundamental da hierarquia SDH tem um ritmo de transmissão de $155{,}52Mb/s$ e designa-se por STM-1 (*Synchronous Transport Module level-1*) em qualquer dos domínios. Níveis de multiplexagem de ordem N são obtidos entrelaçando byte-a-byte N tramas fundamentais. Os ritmos de transmissão são múltiplos exactos do ritmo elementar. As taxas de transmissão STM-N são equivalentes aos ritmos de transmissão STS-3N. A tabela 2.2 mostra os ritmos de transmissão SONET/SDH mais comuns.

Nas normas SONET/SDH, à semelhança do PDH, as tramas têm uma duração de $125\mu s$, ou seja, são transmitidas 8000 tramas por segundo. Isto equivale a dizer que um byte de informação continua a ser um canal de $64Kb/s$.

SONET (Eléctrico)	SONET (Óptico)	Equivalente SDH	Taxa de transmissão (Mb/s)
STS-1	OC-1	-	51,84
STS-3	OC-3	STM-1	155,52
STS-12	OC-12	STM-4	622,08
STS-24	OC-24	STM-8	1 244,16
STS-48	OC-48	STM-16	2 488,32
STS-96	OC-96	STM-32	4 976,64
STS-192	OC-192	STM-64	9 953,28
STS-768	OC-768	STM-256	39 813,12

Tabela 2.2: Taxas de transmissão SDH/SONET

2.2.2 Trama SDH

A norma SDH define uma hierarquia de camadas (ou secções) funcionais da rede de transporte. Cada secção possui um cabeçalho próprio que permite realizar operações de administração e supervisão. A secção de regenerador tem associado o cabeçalho RSOH (*Regenerator Section OverHead*), a secção de multiplexagem está associada ao cabeçalho MSOH (*Multiplex Section OverHead*), enquanto a secção de caminho utiliza o POH (*Path OverHead*). Os contentores virtuais (ou *Virtual Containers* - VC) são as unidades de informação de utilizador e são formados pelo Contentor (*Container* - C) e o cabeçalho de caminho POH. O nome do contentor tem associado um número que indica a sua capacidade, por exemplo C-4 transporta dados ao ritmo de 149,76Mb/s. Os contentores podem estender-se por mais do que uma trama e a posição inicial do VC é indicada pelo ponteiro. Existem ainda alguns bytes extra de justificação, que podem ser usados para compensar diferenças entre o ritmo de transmissão da trama e do VC-4.

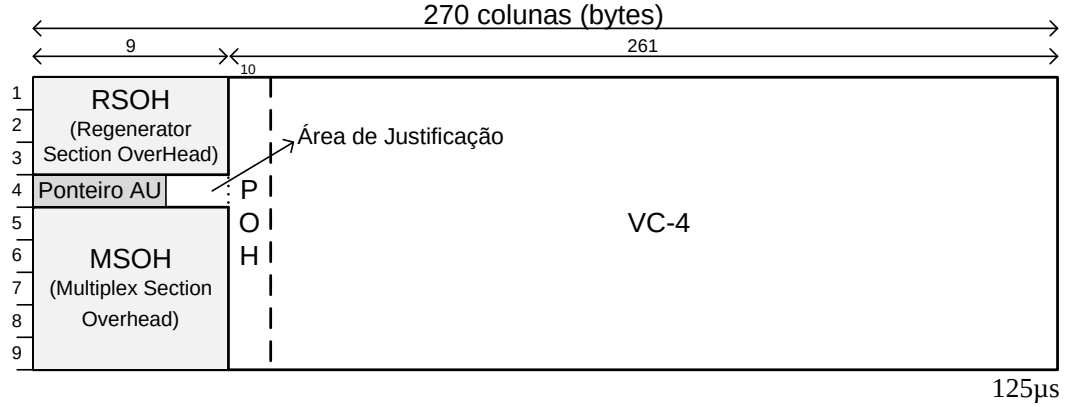


Figura 2.3: Constituição do sinal STM-1 da hierarquia SDH

A representação da trama SDH é normalmente dividida em 9 linhas, de modo a que as primeiras colunas correspondem aos cabeçalhos. A trama é transmitida da esquerda para a direita e de cima para baixo. O número de colunas depende do ritmo de transmissão, por exemplo na figura 2.3 é apresentada a trama STM-1, que corresponde ao primeiro nível da hierarquia SDH, ou seja, 155,52Mb/s. Este sinal possui 9 colunas para os cabeçalhos e 261 para a carga paga. Cada coluna tem o comprimento de um byte, o que perfaz um total de 2430 bytes. Sinais de ritmo mais elevado são obtidos por multiplexagem byte-a-byte de sinais

STM-1, assim sinais STM-N possuem $270N$ colunas.

O Sinal STM-1 pode ser formado a partir de vários tipos de contentores. O STM-1 da figura 2.3 é construído a partir do contentor C-4, que é o equivalente ao sinal E-4 da hierarquia PDH. O cabeçalho de caminho POH é adicionado ao C-4 para formar o contentor virtual de nível 4 (VC-4). A Unidade Administrativa nível 4 AU-4 (*Administrative Unit-level 4*) é composta pelo VC-4 e ainda o ponteiro AU-4. Neste exemplo o AUG (*Administrative Unit Group*) é igual ao AU-4, mas pode ser diferente se forem usados outros contentores. Por exemplo, o AUG pode também ser formado por 3 AU-3 [Lei04]. Por fim são adicionados os cabeçalhos RSOH e MSOH para completar o sinal STM-1.

Bytes do cabeçalho STM-1

Na figura 2.4 é apresentada a disposição dos bytes do cabeçalho STM-1. A função de cada byte é descrita em pormenor na norma G.707 [IT07]. Os mais importantes são os bytes A1 e A2, usados para identificar o início da trama, que têm sempre o mesmo padrão, A1=1111 0110 e A2=0010 1000. À excepção da primeira linha da trama, todos os bytes passam por um *scrambler* que codifica a trama e evita que ocorram longas sequências de '1's ou '0's, permitindo que o sinal de relógio possa ser recuperado correctamente no terminal receptor.

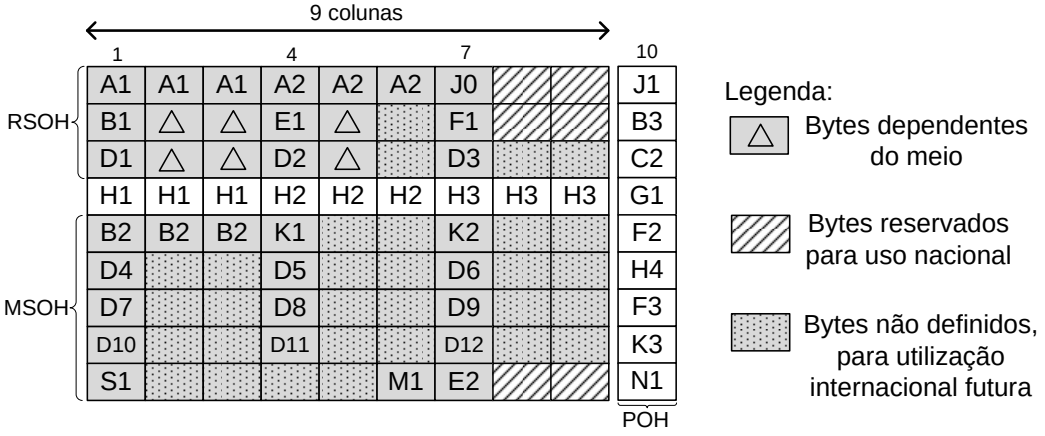


Figura 2.4: Disposição dos bytes do cabeçalho STM-1 [AGI00]

Outros bytes do cabeçalho muito importantes são B1 e B2, usados para detecção de erros na trama. O byte B1 é utilizado para detecção de erros na camada de regenerador, sendo aplicado um código BIP-8 (*Bit Interleaved Parity-8*), com paridade par. Este byte é calculado sobre todos os bytes da trama anterior e colocado na trama actual. Os bytes B2 funcionam de maneira semelhante, mas aplicando BIP-24.

Os ponteiros anteriormente referidos encontram-se nos bytes H1 e H2. No caso do sinal STM-1 transportar um VC-4, apenas um byte H1 e um byte H2 são utilizados, no entanto se forem transportados 3 VC-3, todos os bytes são necessários. Os bytes H3 são usados para justificação negativa.

2.3 Redes Ópticas de Transporte (OTN)

O OTN (*Optical Transport Network*) é definido pelo ITU-T como sendo *um conjunto de elementos de rede ópticos, interligados por fibra óptica, capaz de fornecer funcionalidades de transporte, multiplexagem, routing, gestão e supervisão dos canais ópticos, transportando os sinais provenientes do cliente.* Este novo protocolo para comunicações ópticas pode transportar

diversos tipos de sinais encapsulados na mesma trama. Devido a esta característica é também chamado de invólucro digital (*Digital Wrapper* - DW).

Devido à sua flexibilidade, as redes OTN são particularmente eficientes para tráfego de dados, ao contrário dos antigos SONET/SDH e PDH, mais vocacionados para tráfego de voz. Com a crescente utilização de serviços sobre IP/Ethernet, o OTN torna-se a melhor escolha para transportar este tipo de dados. Outra vantagem relativamente ao SONET/SDH é a introdução de um código de erros, que permite uma maior qualidade de serviço e uma diminuição de custos, pois os regeneradores podem estar mais espaçados.

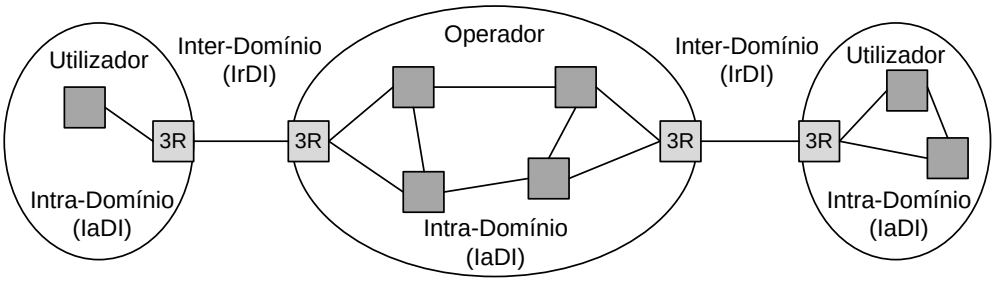


Figura 2.5: Interfaces IrDI e IaDI em redes OTN [Kaz06]

A recomendação G.872 [IT01] define duas classes de interface para redes ópticas de transporte, o *Inter-domain Interface* (IrDI) e o *Intra-domain Interface* (IaDI). A interface IrDI é o processamento 3R (*Retiming, Reshaping and Reamplification*) entre terminais de diferentes operadores. O IaDI é a interface dentro do operador, tal como mostrado na figura 2.5. A norma G.709 é aplicada às interfaces IrDI e IaDI, no entanto apenas define a interface lógica, não especificando a interface óptica nem eléctrica [Kaz06].

2.3.1 Hierarquia (OTN)

A norma OTN permite que múltiplos canais ópticos sejam transportados simultaneamente pela mesma fibra. Cada canal óptico é formado por uma estrutura digital composta pela carga-paga, cabeçalhos e código de erros. Esta estrutura é dividida em várias camadas, formando a hierarquia OTN. O esquema A) da figura 2.6 mostra as várias camadas da hierarquia OTN. As camadas OPU, ODU e OTU são processadas no domínio eléctrico. O OPU (*Optical channel Payload Unit*) engloba o sinal do cliente (que pode ser SDH, SONET, IP, *Gigabit Ethernet...*) e permite pequenos ajustes do ritmo de transmissão. Esta camada é mapeada na fonte, desmapeada no destino e não é alterada pela rede. O ODU (*Optical channel Data Unit*) é formado pelo OPU e um cabeçalho que permite a monitorização dos dados transportados ao longo da rede. O OTU (*Optical channel Transport Unit*) acrescenta o FEC (*Forward Error Correction*) e um pequeno cabeçalho ao ODU. Por último, a estrutura OTU é convertida para o domínio óptico, formando o canal óptico (OCh - *Optical Channel*) [Kaz06]. Cada canal óptico é transportado por um comprimento de onda (OCC - *Optical Channel Carrier*), à semelhança do que acontece no DWDM.

O esquema B) da figura 2.6 mostra a divisão das camadas OTN do ponto de vista da rede de transporte. Como já foi referido, a camada OCh está relacionada com a construção do canal óptico a partir do sinal de cliente e como tal está associada ao processamento no domínio eléctrico. A camada de multiplexagem óptica OMS (*Optical Multiplex Section*) está associada a dispositivos ópticos que fazem encaminhamento de canais no domínio óptico (OADM, OXC). A camada de transmissão óptica OTS (*Optical Transmission Section*) está associada a dispositivos como regeneradores 3R, que permitem o transporte do sinal óptico a

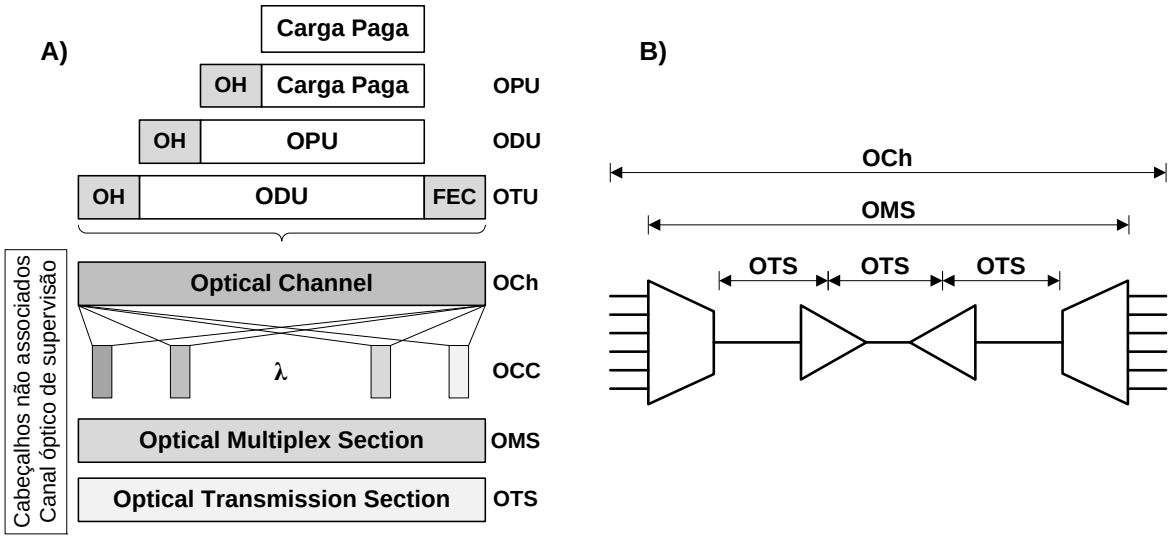


Figura 2.6: Hierarquia OTN conceptual (A) e do ponto de vista da rede de transporte (B)

longas distâncias. Devido à falta de consenso nas implementações, as camadas OTS e OMS ainda não estão definidas por normas do ITU-T. Para efeitos práticos apenas estão definidas 4 camadas, são elas o OPU, ODU, OTU e OCh [Kaz06].

2.3.2 Trama G.709 (OTN)

A figura 2.7 mostra a constituição da trama OTN, como definido na norma do ITU-T G.709. A trama é composta pelos cabeçalhos OTU, ODU e OPU, a carga-paga e os bytes do FEC. A trama é organizada numa estrutura de 4 linhas e 4080 colunas. A transmissão é feita da esquerda para a direita e de cima para baixo.

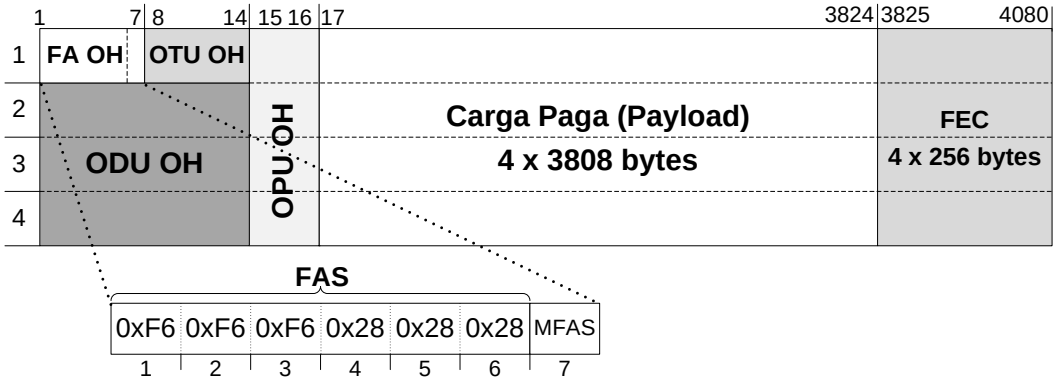


Figura 2.7: Trama OTU composta pelo cabeçalho, carga paga e FEC [IT03b]

O *Forward Error Correction* (FEC) utiliza um código de erros *Reed-Solomon(255,239)* e permite um ganho de codificação considerável. Por exemplo, para um sistema onde se pretenda uma taxa de erros de bit (*Bit Error Rate* - BER) da ordem dos 10^{-15}, a utilização do FEC permite um ganho de $6{,}2dB$ [Wal05]. Isto significa que para uma mesma taxa de erros, a codificação permite reduzir a potência de transmissão em $6{,}2dB$. Por outro lado, significa que os regeneradores podem estar mais espaçados, sem que a taxa de erros aumente. O funcionamento do FEC é apresentado em detalhe mais à frente.

O *Frame Alignment OverHead* (FA OH) está presente em todas as tramas e é composto pelo campo *Frame Alignment Signal* (FAS) e pelo *Multi-Frame Alignment Signal* (MFAS), como mostrado na figura 2.7. O campo FAS é um conjunto de bytes de valor fixo, que são usados para delimitar o início e o final da trama. À semelhança do que acontece no SDH, todos os bits após a sequência FAS passam por um *Scrambler* que evita que existam longas sequências de '0's ou '1's, o que facilita a extracção do sinal de relógio no receptor. Além disso torna menos provável que a sequência FAS se repita ao longo da trama. O FAS é composto por 3 bytes OA1=1111 0110, seguidos por 3 bytes OA2=0010 1000. O campo MFAS é um byte cujo valor é continuamente incrementado trama após trama desde 0 até 255. Este contador é útil em estruturas onde a informação se espalha por várias tramas, como por exemplo o campo PSI (*Payload Structure Identifier*). A organização dos vários campos do cabeçalho OTN é apresentada na figura 2.8. Alguns dos campos descritos são divididos em vários sub-campos.

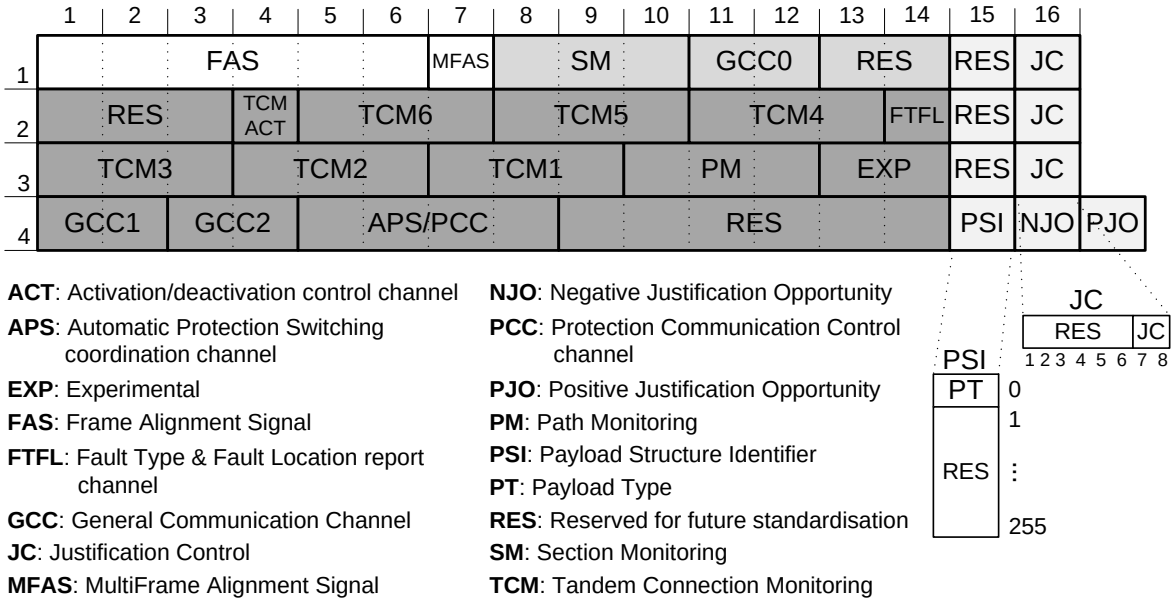


Figura 2.8: Estrutura do cabeçalho de uma trama OTN [IT03b]

Cabeçalho OPU

O cabeçalho OPU é a estrutura utilizada para adaptar a informação do cliente para o transporte no canal óptico. Possui o campo *Payload Structure Identifier* (PSI), que se espalha por 256 tramas. O primeiro byte do PSI é o *Payload Type* (PT), os demais bytes estão reservados para futura utilização. O OPU possui também campos associados ao mapeamento do sinal do cliente dentro da carga útil (JC, PJO e NJO). Os campos PJO e NJO são usados para realizar justificação positiva e negativa, respectivamente. Os bytes JC são constituídos por dois bits de controlo (bits 7 e 8) e 6 bits reservados para utilização futura. Os bits de controlo indicam se os bytes PJO e NJO estão a ser usados para justificação ou dados. Se não ocorrerem erros, os 3 bytes JC devem ser iguais, caso contrário a decisão sobre a função dos bytes de justificação é tomada por maioria (2 em 3) [IT03b].

Cabeçalho OTU

O cabeçalho OTU é composto pelo campo SM (*Section Monitoring*) e por um canal de comunicações genérico GCC0. Os restantes 2 bytes são reservados para utilização futura. O sub-campo TTI (*Trail Trace Identifier*) está presente em várias partes do cabeçalho, incluindo no SM, e indica a origem e o destino da trama. Tem comprimento de 1 byte, no entanto estende-se ao longo de várias tramas, como indicado na figura 2.9. Outro sub-campo do SM comum a outras secções do cabeçalho é o código de detecção de erros BIP-8, que usa paridade par e é idêntico ao byte B1 da trama SDH. O SM possui ainda diversos bits de alarme, como apresentado na figura 2.9.

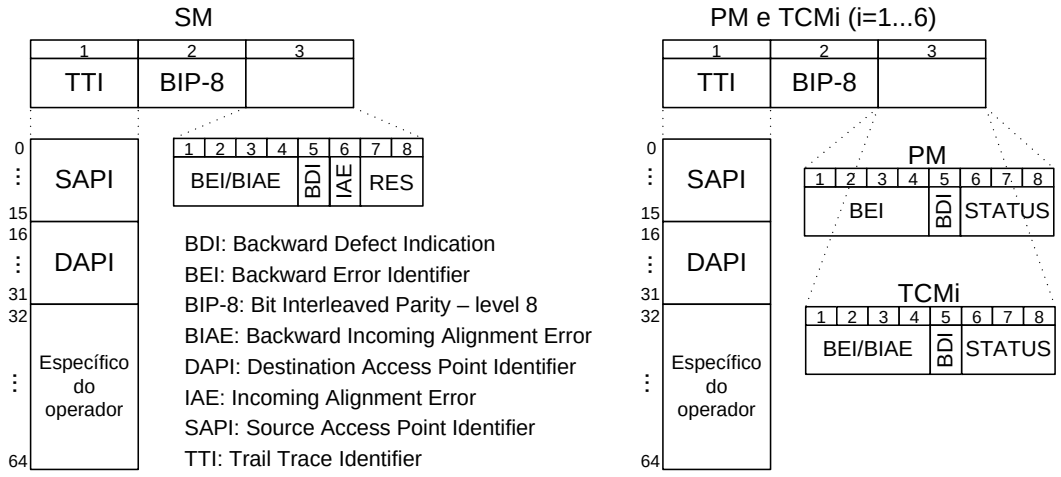


Figura 2.9: Constituição dos campos SM, PM e TCM do cabeçalho OTN [IT03b]

Cabeçalho ODU

Os campos mais importantes do cabeçalho ODU são o PM (*Path Monitoring*) e o TCM (*Tandem Connection Monitoring*), a sua estrutura é apresentada na figura 2.9 e é muito semelhante ao SM. A principal diferença é o campo STATUS que indica a presença de um sinal de manutenção na trama.

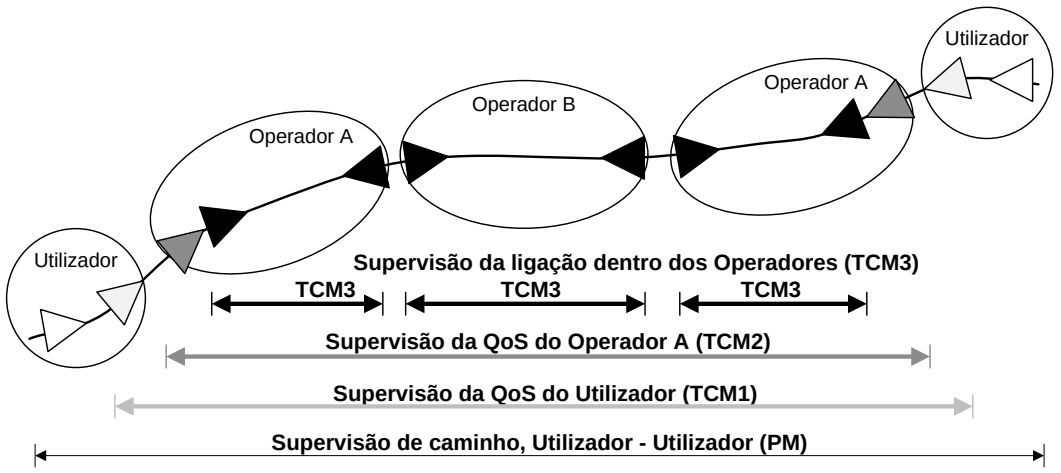


Figura 2.10: Exemplo de utilização do TCM (*Tandem Connection Monitoring*) [Wal05]

O campo TCM é utilizado para supervisionar um segmento do caminho óptico que passa por mais do que um operador de telecomunicações. Os protocolos SONET/SDH foram modificados para permitir uma ligação TCM, já o OTN permite supervisionar até 6 ligações TCM independentes. A figura 2.10 é um exemplo onde a utilização de ligações TCM é útil. Neste exemplo o operador A precisa que o operador B transporte o seu sinal, e por isso A precisa de monitorizar o tráfego que passa por B, o que é conseguido utilizando um ou mais campos TCM do cabeçalho ODU. Ainda na figura 2.10, o TCM1 é usado pelos *Utilizadores* para monitorizar a qualidade de serviço (QoS - *Quality of service*) entre eles, enquanto o TCM2 é usado pelo operador A para monitorizar a sua QoS. Já o TCM3 serve para os operadores supervisionarem o tráfego dentro da sua própria rede [Wal05]. O campo PM (*Path Monitoring*) é utilizado para monitorizar o caminho entre *utilizadores*, independentemente das ligações TCM.

2.3.3 Taxas de transmissão

Ao contrário do que acontece nas normas SONET/SDH, a taxa de transmissão do OTN é aumentada mantendo sempre a mesma estrutura e organização de bytes da trama, ou seja, qualquer trama OTN tem sempre 4 linhas e 4080 colunas. Isto significa que tramas com taxa de transmissão mais elevada têm período menor, em vez de ser constante como nas normas SONET/SDH ($125\mu s$). Na tabela 2.3 são apresentados os 3 ritmos de transmissão definidos na norma ITU-T G.709, que correspondem a OTU-1, OTU-2 e OTU-3. São também indicadas as durações das tramas e os ritmos equivalentes a SONET/SDH. De notar que os ritmos de transmissão OTN são aproximadamente 7% superiores aos equivalentes SONET/SDH, devido aos bytes extra usados para correcção de erros. Actualmente encontra-se em fase de desenvolvimento o ritmo de transmissão OTU-4 que poderá transportar os futuros sinais 100GbE (*Gigabit Ethernet*) [NOR08].

Padrão OTN (G.709)	Ritmo de transmissão	Duração da trama	Equivalente SONET/SDH
OTU-1	$2{,}666Gb/s$	$48{,}971\mu s$	OC-48/STM-16 ($2{,}488Gb/s$)
OTU-2	$10{,}709Gb/s$	$12{,}191\mu s$	OC-192/STM-64 ($9{,}953Gb/s$)
OTU-3	$43{,}018Gb/s$	$3{,}035\mu s$	OC-768/STM-256 ($39{,}813Gb/s$)

Tabela 2.3: Taxas de transmissão OTN e respectivos equivalentes em SONET/SDH

2.4 *Forward Error Correction* do OTN

O *Forward Error Correction* (FEC) especificado na norma OTN usa um código de erros *Reed-Solomon(255,239)*. Para perceber o seu funcionamento é necessário conhecer a teoria por detrás dos códigos de erros em geral e em particular dos códigos *Reed-Solomon*. Nas secções seguintes será abordada a aritmética sobre corpos finitos (ou de Galois), que é a base da teoria da codificação. Serão mencionadas várias técnicas de correcção de erros, focando o estudo nos códigos de blocos, onde se inserem os códigos *Reed-Solomon*. Por fim os códigos *Reed-Solomon* são analisados em detalhe, nomeadamente o seu funcionamento e as técnicas de codificação/descodificação existentes actualmente.

2.4.1 Corpos finitos (ou de Galois)

Os códigos de erros lineares estão definidos sobre estruturas algébricas chamadas corpos finitos ou corpos de Galois. Os corpos onde normalmente se efectuam as operações da aritmética convencional são $\mathbb{Q}$, $\mathbb{R}$ ou $\mathbb{C}$, que constituem corpos com um número infinito de elementos. Formalmente, um corpo é uma estrutura algébrica formada por um conjunto de elementos F e duas operações, $+$ e $\cdot$, geralmente designadas por adição e multiplicação (que podem não coincidir com a adição e multiplicação da aritmética convencional) definidas para todos os pares de elementos de F, onde se verificam as seguintes propriedades:

$$\begin{aligned} \forall x, y, z &\in F \\ x + y &\in F \\ x + (y + z) &= (x + y) + z \\ x + y &= y + x \\ x \cdot y &\in F \\ x \cdot y &= y \cdot x \\ x \cdot (y \cdot z) &= (x \cdot y) \cdot z \\ x \cdot (y + z) &= x \cdot y + x \cdot z \end{aligned}$$

ou seja, um corpo é fechado para as operações de adição e multiplicação e as propriedades de comutatividade, associatividade e distributividade verificam-se [Roc07b].

É possível construir corpos de Galois com q elementos $CG(q)$ se q for da forma $q = P^m$, onde P é um número primo e m é um inteiro positivo [WB94]. Corpos finitos com $q = P$ elementos ($m = 1$) podem ser facilmente construídos utilizando adições e multiplicações módulo P. Uma operação módulo P corresponde ao resto da divisão por P. O corpo finito mais simples é $CG(2)$ cujos elementos são $\{0, 1\}$, sendo as operações efectuadas módulo 2, ou seja, a adição corresponde ao XOR e a multiplicação ao AND.

Quando o número de elementos é uma potência de um número primo, $CG(P^m)$ com $m > 1$, o processo de construção é mais complexo e faz recurso a polinómios definidos sobre $CG(P)$. Na teoria da codificação existe especial interesse sobre corpos finitos de 2^m elementos ($P = 2$).

Corpos finitos de 2^m elementos

A construção de corpos finitos $CG(2^m)$, com $m > 1$, faz uso de polinómios definidos sobre $CG(2)$, de grau m, que são representados por $p(x) = a_m x^m + \cdots + a_1 x + a_0$. Se $a_m = 1$, diz-se que o polinómio é normalizado. Polinómios sobre $CG(P)$ possuem as mesmas propriedades que polinómios sobre $\mathbb{R}$ ou $\mathbb{C}$, podendo ser somados, multiplicados, divididos e até factorizados. Obedecem ainda ao teorema fundamental da álgebra que diz que um polinómio de grau m possui m raízes. Um polinómio irredutível sobre $CG(2)$ é um polinómio que não pode ser factorizado em polinómios sobre $CG(2)$. Este tipo de polinómios é usado para construir corpos $CG(2^m)$, efectuando-se operações módulo polinómio irredutível (à semelhança dos números primos).

Por exemplo para formar o corpo $CG(2^3)$ é usado o polinómio irredutível em $CG(2)$ $p(x) = x^3 + x + 1$. Considerando que o elemento primitivo α é uma raiz de $p(x)$, então $\alpha^3 + \alpha + 1 = 0$ ou $\alpha^3 = \alpha + 1$ [2]. Assim o corpo $CG(8)$ é constituído pelos elementos da tabela 2.4.

[2] Em $CG(2)$ a subtracção é equivalente à adição.

Representação Exponencial	Representação Polinomial	Representação vectorial (binária)	Representação vectorial (decimal)
0	0	000	0
1	1	001	1
α^1	α	010	2
α^2	α^2	100	4
α^3	$\alpha + 1$	011	3
α^4	$\alpha^2 + \alpha$	110	6
α^5	$\alpha^3 \cdot \alpha^2 = \alpha^2 + \alpha + 1$	111	7
α^6	$\alpha^3 \cdot \alpha^3 = \alpha^2 + 1$	101	5

Tabela 2.4: Representação dos elementos de $CG(8)$ [WB94] [Moo05]

A representação exponencial, tabela 2.4, é útil para efectuar multiplicações. Para calcular $\alpha^a \alpha^b = \alpha^c$ em $CG(2^m)$ basta determinar c que é dado por [WB94]

$$c = (a + b) \quad mod \quad (2^m - 1)$$

Já para somar é preferível usar a notação polinomial. Por exemplo para calcular $\alpha^2 + \alpha^5$ basta substituir pelos polinómios equivalentes e obtém-se

$$\alpha^2 + \alpha^5 = (\alpha^2) + (\alpha^2 + \alpha + 1) = (\alpha + 1) = \alpha^3$$

No entanto para corpos finitos do tipo $CG(2^m)$ a adição pode ser implementada de modo mais simples, recorrendo à representação binária, pois corresponde a fazer um XOR bit-a-bit dos operandos.

2.4.2 Códigos correctores de erros genéricos

Para tornar o canal mais imune a imperfeições, é normal utilizar algum tipo de codificação da informação. As técnicas de codificação baseiam-se na introdução de redundância, ou seja, acrescentar algo mais à mensagem a transmitir. Se esse acréscimo de informação for escolhido adequadamente tal pode resultar em vantagens consideráveis na fiabilidade da transmissão. A informação redundante é usada para detectar e/ou corrigir erros da mensagem recebida, assim a desvantagem de transmitir mais informação é compensada pela maior qualidade da comunicação. A capacidade de correcção de erros pode também ser usada para transmitir uma mensagem a maior distância, usando a mesma potência e mantendo a taxa de erros.

Redundância e o limite de Shannon

Em comunicações digitais qualquer mensagem M de duração finita é constituída por uma combinação finita de símbolos de um alfabeto restrito. No caso de comunicações digitais binárias é usado o alfabeto $\{0, 1\}$ e qualquer mensagem binária de duração NT, onde T é o ritmo de transmissão, consiste numa combinação de N símbolos binários (bits). O número total de combinações possíveis é 2^N e tem-se redundância se para formar as mensagens M for utilizado apenas um subconjunto dessas 2^N possibilidades. Se a mensagem M recebida não pertencer ao subconjunto predefinido, significa obrigatoriamente que ocorreram erros durante a transmissão. Sem redundância qualquer combinação dos símbolos do alfabeto é tão válida como qualquer outra, não sendo possível detectar ou corrigir erros. É também claro que mesmo havendo redundância, nem todos os tipos de erros podem ser detectados. Se um

erro transformar uma mensagem transmitida noutra mensagem válida, não é possível detectar os erros [Roc07a].

A introdução de bits de redundância à mensagem não codificada implica que a taxa de transmissão seja superior, para o mesmo período de transmissão. No entanto uma taxa de transmissão superior requer uma maior largura de banda, o que pode provocar um aumento da taxa de erros caso haja limitações de banda. Shannon demonstrou que uma codificação adequada compensa o aumento da probabilidade de erro causada pelo aumento do ritmo de transmissão [Sha48]. O teorema de Shannon veio revolucionar o problema da comunicação, dizendo que *o ruído não é um factor limitativo da qualidade da comunicação mas sim da quantidade máxima de informação que pode ser transmitida com fidelidade.*

O limite de Shannon é dado por

$$\frac{E_b}{N_o} > \ln(2)(-1{,}59dB)$$

e significa que mesmo utilizando uma largura de banda infinitamente grande, consegue-se uma transmissão confiável desde que seja garantida uma relação sinal ruído de pelo menos -1,59dB [BC02]. Isto é conseguido se for utilizada uma codificação adequada. No entanto a demonstração de Shannon não define o tipo de codificação. Ao longo dos anos muita investigação tem sido desenvolvida na tentativa de encontrar um código que permita um desempenho próximo do limite teórico de Shannon.

Técnicas de controlo de erros

As técnicas de controlo de erros em comunicação consistem essencialmente em acrescentar redundância controlada às mensagens transmitidas de modo que mesmo que parte da informação seja perdida ou alterada, seja possível detectar e eventualmente corrigir os erros ocorridos. As técnicas de controlo de erros podem dividir-se em 3 classes fundamentais [Roc07a]:

- **FEC (Forward Error Correction)** - Corresponde à inclusão de redundância no dicionário de modo a permitir não só detectar como corrigir erros.
- **ARQ (Automatic Repeat Request)** - Se um canal de retorno estiver disponível, após a detecção da ocorrência de erros pede-se ao emissor uma retransmissão da mensagem.
- **Técnicas híbridas FEC-ARQ** - O receptor tem capacidade de correcção de um certo número de erros, se o número de erros exceder a capacidade de correcção, é pedido ao emissor que reenvie a mensagem.

As técnicas baseadas em ARQ são mais simples de implementar, pois apenas necessitam de detectar erros, mas é necessário um canal de retorno para pedir a retransmissão. Por sua vez, o FEC necessita de códigos com capacidade de correcção e logo mais complexos de implementar, no entanto estes têm muito mais interesse prático. Os códigos FEC podem ser divididos em códigos de blocos e códigos convolucionais. Nas secções seguintes serão abordados os códigos de blocos em geral e em particular os códigos Reed-Solomon.

2.4.3 Códigos de blocos

Num código de blocos, a informação é dividida em palavras de k símbolos que são mapeados em blocos independentes de n símbolos ($n > k$). Assim cada palavra de código tem $n - k$ símbolos de paridade. Caso o código seja binário os símbolos são os bits $\{0, 1\}$, podendo ser formados códigos com um conjunto arbitrário de símbolos. No entanto em sistemas digitais

têm mais interesse códigos em que o número de símbolos é uma potência de 2, pois deste modo cada símbolo pode ser representado pelo equivalente binário, sem que haja representações binárias não usadas. O nome de um código de blocos é escrito na forma: *código(n,k).*

Para definir a capacidade de detecção/correcção de um código de blocos é normal utilizar o parâmetro *distância de Hamming do código* que é definido como sendo a *distância de Hamming* mínima entre duas palavras do código. A *distância de Hamming* entre duas palavras é o número de posições em que estas diferem. Se um código tiver distância d então pode ser usado para:

- Detectar até $d-1$ erros;
- Corrigir até $\lfloor \frac{d-1}{2} \rfloor$ erros [3].

Códigos de blocos cíclicos

Os códigos cíclicos são códigos de blocos lineares que, para além das propriedades gerais que definem a linearidade, verificam a seguinte propriedade adicional: *Qualquer deslocamento cíclico de uma palavra de código também é palavra de código.*

É usual em códigos cíclicos designar uma palavra de código $(c_0, c_1, \cdots, c_{n-2}, c_{n-1})$ por um polinómio com coeficientes sobre o corpo onde está definido o código:

$$(c_0, c_1, \cdots, c_{n-2}, c_{n-1}) \equiv c(x) = c_{n-1}x^{n-1} + c_{n-2}x^{n-2} \cdots + c_1 x + c_0$$

Em termos polinomiais um deslocamento cíclico para a direita de uma determinada sequência $(c_0, c_1, \cdots, c_{n-2}, c_{n-1})$ significa efectuar a operação $R_{x^{n-1}}(xc(x))$ [4].

Se $g(x)$ for o polinómio normalizado de menor grau pertencente ao código cíclico, então $g(x)$ é o polinómio gerador do código. O grau de $g(x)$ é igual ao número de símbolos de paridade e qualquer palavra do código é um múltiplo de $g(x)$. Um código cíclico é pois o conjunto das combinações lineares de $g(x)$ e dos seus $k-1$ deslocamentos para a direita. O polinómio gerador, não pode no entanto ser escolhido arbitrariamente, $g(x)$ tem de ser um factor de x^{n-1} [Roc07a].

Códigos cíclicos sistemáticos

Um código linear (n, k) diz-se sistemático, se na palavra de código de n símbolos os primeiros (ou os últimos) k elementos são os símbolos da palavra de informação e os restantes $n-k$ elementos são os símbolos redundantes de paridade.

O método de codificação mais simples corresponde a multiplicar a palavra de informação (tamanho k) pelo polinómio gerador (tamanho $n-k$), obtendo-se a palavra de código (tamanho n). No entanto este método gera palavras de código não sistemáticas. Do ponto de vista de implementação os códigos sistemáticos são preferíveis. O modo de obtenção de códigos sistemáticos pode ser descrito pelos seguintes passos [Roc07a]:

1. Multiplicar o polinómio de informação $i(x)$ por x^{n-k};
2. Dividir o resultado de 1. por $g(x)$ e guardar o resto $r(x)$;
3. Subtrair o resto obtido em 2. ao resultado de 1. para obter a palavra de código:

$$c(x) = x^{n-k}i(x) - R_{g(x)}(x^{n-k}i(x))$$

[3] $\lfloor x \rfloor$ representa o maior inteiro menor ou igual a x.
[4] $R_{p(x)}(a(x))$ designa o resto da divisão polinomial de $a(x)$ por $p(x)$.

2.4.4 Códigos Reed-Solomon

Códigos *Reed-Solomon* [5] (RS) são códigos de blocos, usados normalmente de forma sistemática. Além disso, são códigos lineares (a adição de duas palavras de código é também uma palavra de código) e são cíclicos (um deslocamento cíclico dos símbolos de uma palavra de código produz outra palavra código). Pertencem à família dos códigos BCH [6], mas distinguem-se destes por usarem símbolos multi-bit. Esta característica faz dos códigos Reed-Solomon particularmente eficientes para lidar com longas sequências de bits errados porque vários bits errados num símbolo contam apenas como um erro de símbolo, para efeitos de capacidade de correcção [Cla02].

Um código $RS(n,k)$ com símbolos de m bits é construído sobre $CG(2^m)$. O valor n depende do tamanho dos símbolos, segundo a relação $n = 2^m - 1$. Valores de $n < 2^m - 1$ são usados em versões "encurtadas" de códigos RS. Se for pretendido que o código corrija t símbolos errados, então k tem que verificar a relação $t = \frac{(n-k)}{2}$.

A distância mínima dos códigos $RS(n,k)$ é $d = n - k + 1$, que é o limite superior de distância de um código (n,k). Isto significa que não existe nenhum código de blocos (n,k) com distância de Hamming superior à de um código $RS(n,k)$. Um código com distância d pode corrigir até $\lfloor \frac{d-1}{2} \rfloor$ erros o que é coerente com o parâmetro t definido anteriormente.

A construção do código propriamente dito recorre a um polinómio gerador $G(x)$ que tem de ter como raízes $2t$ potências consecutivas de α (elemento primitivo do corpo), ou seja,

$$G(x) = \prod_{i=b}^{2t-b-1} (x - \alpha^i) \tag{2.1}$$

Na prática o sinal menos da equação equivale a uma adição e α é normalmente igual a 2 (tabela 2.4). O valor b é normalmente 0 ou 1, podendo tomar outros valores. Tendo isto em conta, os coeficientes do polinómio podem ser determinados recorrendo a adições e multiplicações sobre $CG(2^m)$.

Aplicações de códigos Reed-solomon

Como são códigos com símbolos multi-bit, os códigos Reed-Solomon são ideais para aplicações onde podem ocorrer grandes sequências de bits errados. Para situações onde erros pontuais são predominantes, haverá outras soluções mais adequadas. As aplicações comerciais mais conhecidas deste tipo de códigos são discos ópticos (CD's, DVD's e Blu-Ray), tecnologias de transferência de dados (DSL, WiMax), transmissões de televisão digital (DVB, ATSC) entre outras. Mais recentemente os códigos RS passaram a ser usados em telecomunicações, como é o caso da norma OTN. Um dos principais problemas destes códigos é a complexidade de implementação, principalmente no que respeita ao bloco de descodificação. Nos últimos anos têm surgido novas técnicas que permitiram reduzir substancialmente os recursos necessários para a implementação.

A primeira aplicação em massa de códigos Reed-Solomon surgiu 1982, quando foi usado no CD. Neste caso o código é capaz de corrigir uma sequência de até 4000 bits (equivalente a um risco de $2,5mm$ na superfície do CD). Isto é conseguido através do entrelaçamento de palavras de código [WB94]. Deste modo uma sequência de erros é repartida por várias palavras de código, o que possibilita a correcção. Esta técnica de entrelaçamento é também usada na recomendação OTN.

[5] Os códigos Reed-Solomon foram inventados em 1960 por Irving S. Reed e Gustave Solomon.

[6] Os códigos BCH foram inventados por Hocquenghem (1959) e independentemente por Bose e Ray-Chaudhuri (1960).

Uma aplicação menos evidente de códigos Reed-Solomon são os sistemas de telecomunicações no espaço, sendo usados pela NASA e ESA. Neste caso o canal de comunicação não é caracterizado por longas sequências de bits errados, no entanto estudos mostram que códigos RS usados em conjunto com códigos convolucionais permitem um grande ganho de codificação. A figura 2.11 mostra a taxa de erros em função da relação sinal ruído para diversos tipos de codificação em comunicações no espaço, tendo como referência o limite teórico (*limite de Shannon*). Ainda na figura 2.11 são mostradas as curvas para sistemas sem codificação, codificação RS não concatenada e codificação RS em conjunto com códigos convolucionais (códigos concatenados).

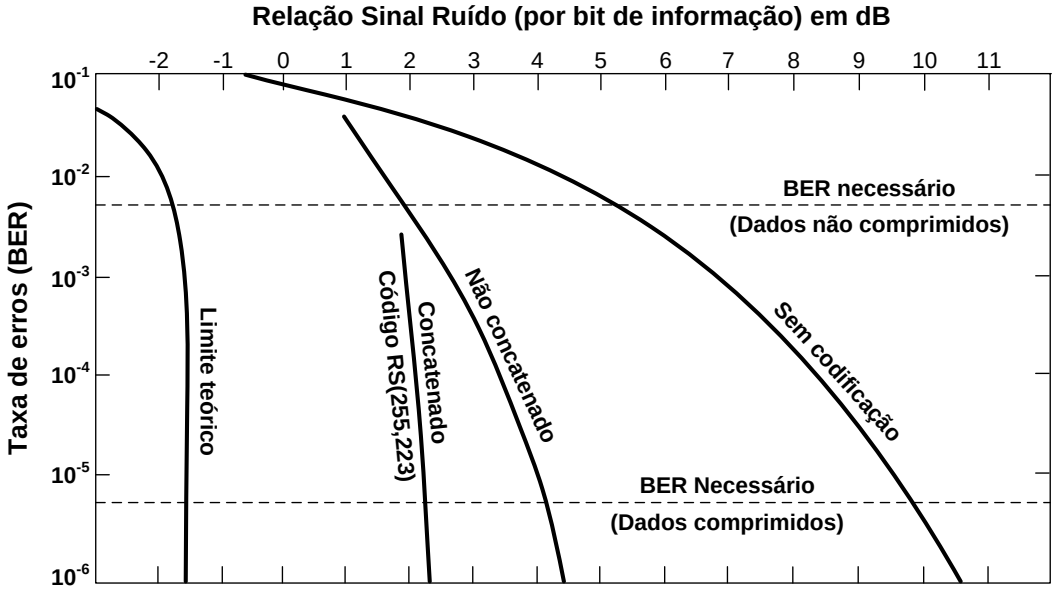


Figura 2.11: Desempenho de vários tipos de codificação em comunicações no espaço [WB94]

A dimensão de um código *Reed-Solomon(n,k)* "normal" não é arbitrária. De facto a construção de um código RS segue regras rígidas, por exemplo $n = 2^m - 1$. Os códigos RS mais comuns têm $n = 255$. Deste modo os símbolos pertencem a $CG(256)$ e correspondem a um byte de informação. Esta restrição em termos de dimensão do código pode ser contornada usando versões encurtadas de códigos RS. Por exemplo, o código $RS(255, 223)$ pode ser encurtado para um código $(160, 128)$. Para a codificação, os 95 símbolos restantes são colocados a zero, mas não são transmitidos. No descodificador a mesma quantidade de zeros é adicionada à palavra recebida. Esta técnica permite usar símbolos de 8 bits para qualquer tamanho da palavra, o que inicialmente estaria limitado a palavras de 255 símbolos. Este método é aplicado, por exemplo, nos CD's.

Reed-Solomon(255,239) do OTN

O *Forward Error Correction* (FEC) definido na norma OTN usa um código de erros *Reed-Solomon(255,239)* com entrelaçamento de símbolos. A designação $RS(255, 239)$ indica que cada palavra de código é constituída por 255 símbolos de 8 bits, sendo que 239 são dados e os restantes 16 são símbolos redundantes de paridade. O facto de o código ser sistemático significa que as operações de codificação e descodificação não alteram os 239 símbolos de dados de cada palavra de código, actuando apenas sobre os símbolos de paridade.

A trama OTN é constituída por 4 linhas, cada uma com 4080 bytes. Para o processamento do FEC cada linha é dividida em 16 sublinhas (palavras de código). No total cada trama é

dividida em 64 palavras de código, que são processadas independentemente. Cada palavra de código é formada por entrelaçamento de bytes segundo o esquema da figura 2.12.

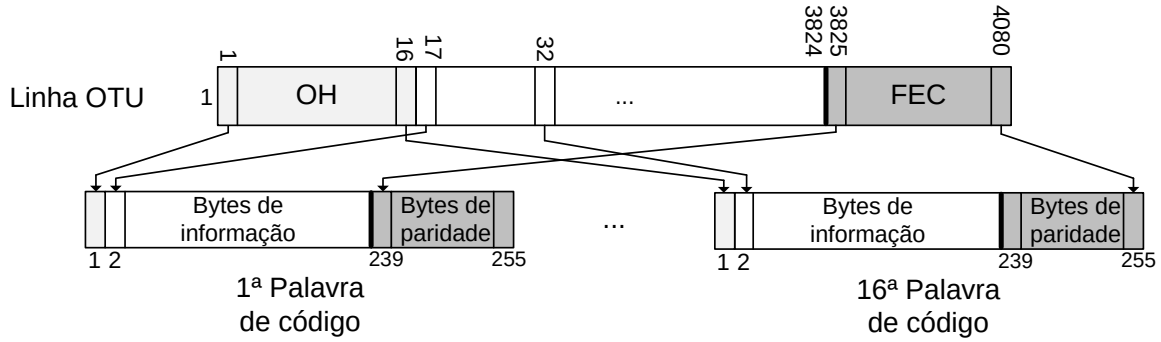


Figura 2.12: Entrelaçamento das palavras de código do OTN [JDS08].

Isto significa que cada uma das 64 palavras de código é composta por um byte do cabeçalho e 16 bytes do FEC, sendo os restantes pertencentes à carga paga. Os bytes de uma linha da trama pertencem à palavra de código X se obedecerem à relação: $X + 16(i-1)$, para $i = (1, 2, \ldots, 255)$.

A distância de Hamming do código $RS(255, 239)$ é $d_{min} = 17$, o que significa que permite corrigir no máximo $\lfloor \frac{dmin-1}{2} \rfloor = 8$ símbolos errados ou detectar até $d_{min} - 1 = 16$ símbolos errados em cada uma das 64 palavras de código. Além disso o entrelaçamento permite aumentar a eficiência, pois deste modo o efeito de uma grande sequência de erros é repartido por várias palavras.

O entrelaçamento permite, por exemplo, corrigir qualquer sequência de 128 símbolos (1024 bits) errados consecutivos numa linha da trama, desde que nas restantes posições dessa linha não haja mais erros [IT00]. Pode também corrigir algumas sequências de erros maiores, até ao máximo de 256 símbolos errados consecutivos (128 no final de uma linha mais 128 no início da seguinte). Desta forma o $RS(255, 239)$ pode corrigir grande parte das combinações de símbolos errados ao longo de uma trama OTU, desde que o total de erros numa trama seja menor que 512.

Como o código usa símbolos de 8 bits, é construído sobre $CG(2^8) = CG(256)$. O corpo de Galois é construído com base no polinómio irredutível sobre $CG(2)$:

$$H(x) = x^8 + x^4 + x^3 + x^2 + 1. \tag{2.2}$$

O polinómio gerador do código, $G(x)$, é definido na norma G.709 pela expressão 2.1, onde $t = 8$ e $b = 0$, ou seja:

$$G(x) = \prod_{i=0}^{15} (x - \alpha^i).$$

O processo de codificação do $RS(255, 239)$ é o método geral usado para qualquer código cíclico sistemático (secção 2.4.3). De acordo com a figura 2.13, o polinómio de dados pode ser representado na forma polinomial por $D(x) = D_{254}x^{254} + D_{253}x^{253} + \ldots + D_{16}x^{16}$. O polinómio dos símbolos de paridade é então dado por

$$R(x) = D(x) \quad mod \quad G(x), \tag{2.3}$$

onde $R(x)$ será da forma $R(x) = R_{15}x^{15} + R_{14}x^{14} + \cdots + R_1x + R_0$.

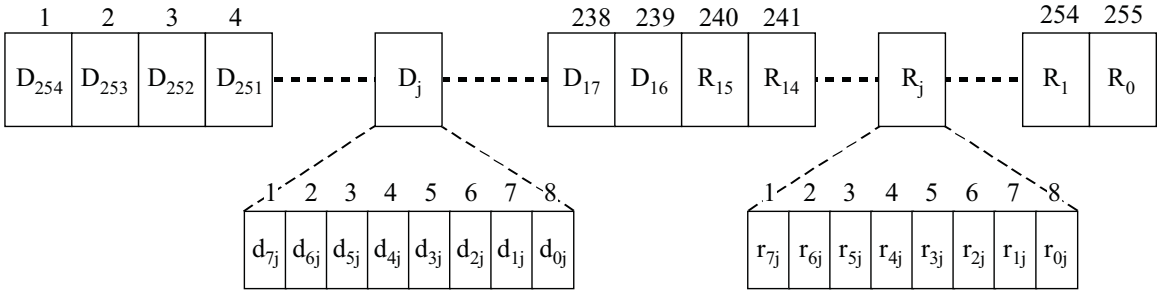


Figura 2.13: Formato de uma palavra de código do FEC [IT03b]

Técnicas de descodificação

Para implementar um sistema de correcção de erros *Reed-Solomon*, o bloco mais complexo é o descodificador. As técnicas gerais de descodificação baseiam-se sobretudo em tabelas que estabelecem a correspondência entre síndroma e palavra de erro. Este método apesar de simples só é possível de usar para códigos de pequena dimensão. Para os códigos RS mais comuns é necessário utilizar uma descodificação algébrica.

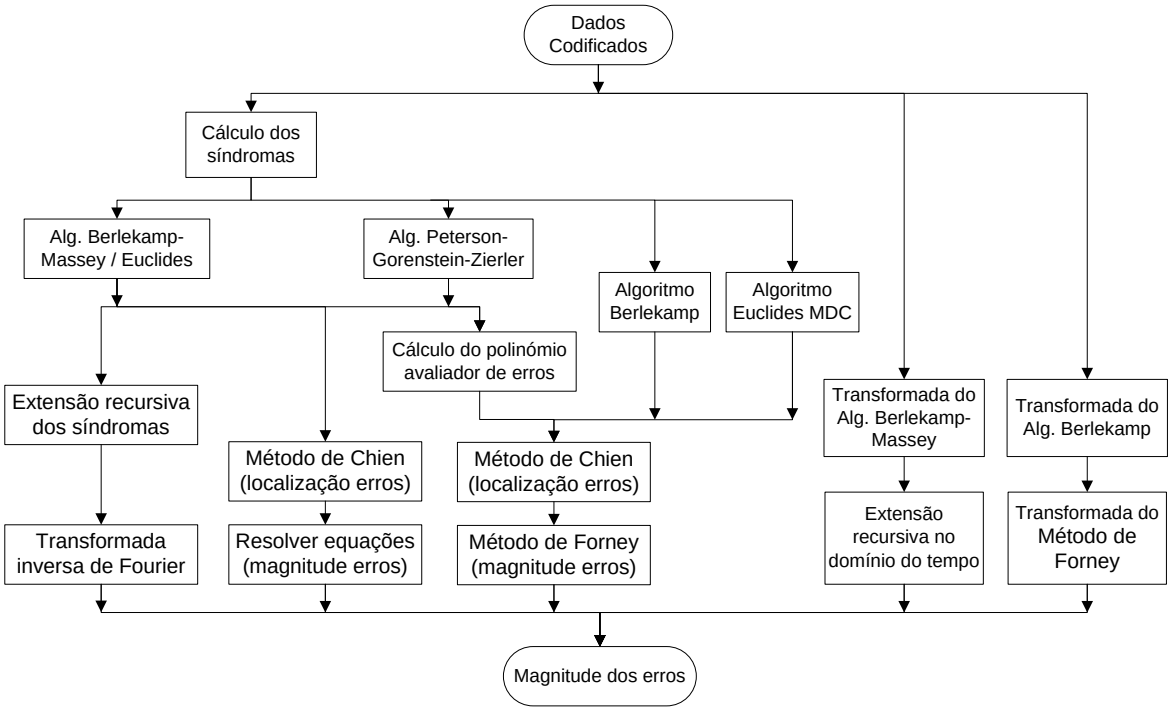


Figura 2.14: Resumo das técnicas de descodificação de códigos RS [RL00]

A figura 2.14 resume os algoritmos de descodificação algébrica de códigos RS. Do lado esquerdo da figura 2.14 estão representados os algoritmos que envolvem o cálculo dos síndromas. Estes serão analisados em primeiro lugar. Após o cálculo dos síndromas, o próximo passo é determinar o polinómio localizador de erros $\Lambda(x)$. A primeira solução a este problema foi proposta por Peterson e melhorada por Gorenstein e Zierler (algoritmo Peterson - Gorenstein - Zierler). Este algoritmo baseia-se na resolução de um sistema de equações envolvendo o polinómio $\Lambda(x)$ desconhecido e os síndromas já conhecidos. A solução do sistema de equações é obtida através de técnicas de inversão de matrizes, o que torna este algoritmo

computacionalmente ineficiente, para códigos com elevada capacidade de correcção [RL00].

Uma técnica eficiente para determinar a localização dos erros foi proposta pela primeira vez por *Berlekamp*. *Massey* adaptou o algoritmo para ser implementado usando um *Linear Feedback Shift Register* (LFSR), dando origem ao algoritmo *Berlekamp-Massey*. Mais tarde, um grupo de investigação mostrou que também é possível usar o algoritmo de *Euclides* do Máximo Divisor Comum (MDC) para determinar $\Lambda(x)$. O passo seguinte é determinar a magnitude dos erros, que pode ser calculada no domínio do tempo ou da frequência. No domínio da frequência, as técnicas são baseadas na transformada inversa de Fourier. No domínio do tempo é usando o método de *Chien* para determinar as raízes do polinómio $\Lambda(x)$ e de seguida a magnitude dos erros é calculada resolvendo directamente um sistema de equações, ou usando o método de *Forney*. O método de *Forney* requer o cálculo do polinómio avaliador de erros $\Omega(x)$, que pode ser obtido resolvendo a Equação-Chave (*Key-Equation*) [RL00].

Outra abordagem à descodificação de códigos RS, apresentada do lado direito da figura 2.14, evita o cálculo dos síndromas e baseia-se na transformada inversa de Fourier. Estes métodos usam a transformada dos algoritmos *Berlekamp-Massey* e *Berlekamp*, por isso todas as variáveis estão no domínio do tempo. A complexidade destas técnicas continua a ser superior às técnicas baseadas no cálculo dos síndromas [RL00].

As implementações mais comuns actuais baseiam-se nos algoritmos *Berlekamp-Massey* ou no algoritmo de *Euclides*. O algoritmo *Berlekamp-Massey* é mais complexo de implementar mas mais eficiente, já o de *Euclides* é mais simples mas menos eficiente. Os principais avanços tecnológicos na área da codificação são sobretudo tentativas de optimizar estes algoritmos. Neste tipo de implementações, os símbolos recebidos são transformados para o domínio da frequência devido ao cálculo dos síndromas, de seguida a localização e magnitude dos erros são determinadas no domínio do tempo com base nos síndromas já conhecidos. Devido a esta mistura de frequência e tempo, são chamados de algoritmos de descodificação híbridos [Bla83].

Exemplos de implementações em hardware usando o algoritmo de *Berlekamp-Massey* são apresentados em [RL00], [MG02]. Outras versões modificadas podem ser encontradas em [SS01], [RST91]. Implementações usando o algoritmo de *Euclides* são apresentadas em [Cla02], [WY05]. Os blocos de cálculo dos síndromas, método de *Chien* e método de *Forney* são descritos detalhadamente em [WY05], [Lee05]. Descodificadores baseados na transformada inversa de *Fourier* são analisados em [WB94], [Moo05].

Códigos de erros modernos

Actualmente os códigos de erros que mais se aproximam do limite teórico de *Shannon* são os *Turbo-codes* e os códigos LDPC (*Low-Density Parity-Check*). Os *Turbo-codes* foram propostos pela primeira vez em 1993 [BGT93] e usam um processo iterativo de descodificação. Após várias iterações, a eficiência do código aproxima-se significativamente do limite teórico [Moo05].

Os códigos LDPC foram descobertos em 1963, mas devido à complexidade de implementação só a partir de 1996 começaram a ser usados. Actualmente os LDPC superam os *Turbo-codes* e são usados em algumas normas como IEEE 802.16, IEEE 802.20, IEEE 802.3 e DBV-RS2 [Moo05]. Tanto os *Turbo-codes* como os LDPC são usados principalmente em aplicações que requerem alto desempenho, como comunicações via satélite e são mais indicados para canais de comunicação que normalmente causam erros pontuais.

Devido à sua grande popularidade, os códigos *Reed-Solomon* continuam a ser bastante usados, principalmente em aplicações onde podem surgir longas sequências de bits errados. Para melhorar o desempenho, um código RS pode ser combinado com outro código de erros, formando os chamados códigos concatenados. Nestes códigos, é frequentemente usado um código convolucional simples em conjunto com o código *Reed-Solomon*, mais complexo. Nestas

situações, além do descodificador RS é também necessário um descodificador *Viterbi* (para o código convolucional). Esta técnica permite aumentar o ganho de codificação [BDM81].

2.5 Implementações da norma G.709

Apesar da norma G.709 (OTN) ser relativamente recente, existem já diversos trabalhos sobre este assunto, nomeadamente artigos a descrever a funcionamento teórico da norma. A tese [Che06] aborda a evolução das redes OTN e o seu futuro, analisando alguns casos práticos, no entanto fica-se pela análise teórica. O presente texto pretende ir mais longe e analisar aspectos relacionados com a implementação da norma.

Em [PM07] é descrita uma implementação em FPGA da norma OTN, no entanto não é feita qualquer monitorização de tráfego, limitando-se a guardar em memória os cabeçalhos recebidos. Além disso, o descodificador FEC não é implementado.

Quanto ao descodificador FEC existem inúmeros artigos sobre implementações em hardware, mas em geral cada artigo dedica-se apenas a uma parte do descodificador. Neste trabalho pretende-se implementar um descodificador completamente funcional, tentando encontrar a melhor solução para a implementação de cada bloco, à semelhança do que é feito na tese [SR07] dedicada exclusivamente ao descodificador FEC.

2.6 Conclusão

Neste capítulo é feita uma introdução aos sistemas de comunicação ópticos, referindo os principais protocolos existentes, nomeadamente os protocolos PDH, SONET, SDH, WDM e OTN. A norma OTN, sendo o tema principal deste projecto, foi abordada em mais detalhe. Sobre o OTN foram referidas as principais características e vantagens relativamente a outros protocolos de rede ópticos. Na análise da norma OTN é descrito o funcionamento da hierarquia, a constituição da trama e as taxas de transmissão.

Um dos principais melhoramentos introduzidos pela norma OTN é o FEC, que usa um código de erros $Reed - Solomon(255, 239)$. Como tal, é dedicada uma grande parte deste capítulo à análise dos códigos de erros. Começando pela teoria dos Corpos Finitos (ou de Galois), são abordados os tipos de códigos de erros mais comuns, focando o estudo nos códigos de blocos e em particular nos códigos *Reed-Solomon*. Sobre estes últimos é descrito o funcionamento geral, referindo algumas aplicações. Foram também referidas algumas técnicas de descodificação existentes e feitas algumas referências a implementações práticas. A teoria descrita neste capítulo será útil para implementar o codificador e descodificador $RS(255, 239)$.

Capítulo 3

Especificação dos blocos

Para sistemas digitais onde os requisitos temporais são muito exigentes e o processamento é intensivo, não é possível usar um processador convencional. Uma alternativa é criar um circuito de *hardware* dedicado para realizar as tarefas pretendidas, ou seja, utilizar um ASIC (*Application-Specific Integrated Circuit*). No entanto este tipo de circuitos tem um tempo de desenvolvimento muito grande, pois as fases do projecto vão desde a especificação até à fabricação do *chip*.

Uma alternativa ao ASIC é o FPGA (*Field Programmable Gate Array*), que é um dispositivo de *hardware* reconfigurável. Os FPGAs são *chips* cuja funcionalidade pode ser configurada por *software*, o que permite desenvolver e testar várias implementações num curto espaço de tempo. O método mais comum de especificar o sistema digital é utilizar uma linguagem de descrição de *hardware*, neste caso o VHDL. O código criado é processado por uma ferramenta de síntese, neste caso a ferramenta ISE da *Xilinx*, que se encarrega de criar o ficheiro de configuração para o FPGA. O código pode ser simulado, neste caso no *ModelSim*, para verificar se o comportamento do circuito implementado corresponde ao pretendido.

A implementação e teste em *hardware* não fazem parte deste projecto, mas no futuro o sistema poderá ser implementado em FPGA. Deste modo, a especificação do sistema tem em conta as restrições temporais impostas por um FPGA real. Na secção seguinte são indicadas as características principais do FPGA que foi escolhido para implementar o sistema. Noutra secção são apresentados os principais cuidados a ter na implementação de circuitos sequenciais e algumas particularidades do implementação em VHDL. As restantes secções deste capítulo são dedicadas à especificação dos diversos blocos que compõem o receptor OTU1.

3.1 Características do dispositivo de hardware (FPGA)

Tendo em conta as exigências temporais e também a área ocupada pelo sistema, é necessário um FPGA que permita um alto desempenho e que por outro lado tenha uma grande quantidade de blocos lógicos e memória disponível. É também desejável que o consumo de energia seja baixo. O dispositivo escolhido é o modelo LX330T pertencente à família de FPGAs *Virtex-5* LXT [Xil07] da *Xilinx*. Este dispositivo é optimizado para implementar lógica com alto desempenho e baixo consumo de energia, para ligações série. As suas principais características são [Xil09a]:

- Frequências de operação até 550MHz;
- 6 *Clocking & Management Tiles* (CMT) para gestão dos sinais de relógio;
- *Look-Up Tables* com 6 entradas (6-LUT);

- 25 920 *Configurable Logic Blocks* (CLBs) com 8 LUTs/Flip-Flops cada;
- Bloco de memória RAM dedicada de 11 664 Kbits;
- Memória RAM distribuída de 3 420 Kbits;
- 24 *transceivers RocketIO* GPT até 3,2Gb/s (consumo de 100mW por cada *transceiver*);

Os dispositivos *Virtex-5* têm uma distribuição do sinal de relógio melhorada, devido ao uso de regiões de relógio. Cada região pode ter até 10 domínios de relógio globais. Na figura 3.1 é apresentada a distribuição das 24 regiões de relógio do dispositivo LX330T. Cada região possui uma altura de 20 CLBs (*Configurable Logic Blocks*) e estende-se em comprimento por metade do *chip* [Xil09c].

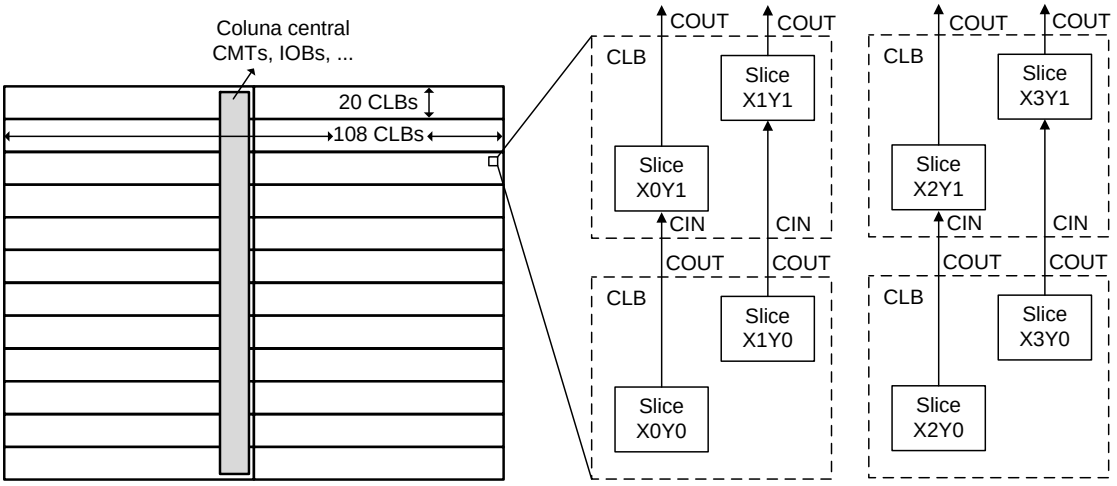


Figura 3.1: Regiões de relógio do dispositivo LX330T e distribuição de CLBs

Cada CLB é formado por 2 *slices*: 1 SLICEL + 1 SLICEM ou 2 SLICEL. Cada *slice* é composto por 4 *Look-Up Tables* (LUTs) e 4 *Flip-Flops*. Cada LUT pode implementar funções lógicas com um máximo de 6 entradas (6-LUTs). Os SLICEM têm ainda a capacidade adicional de implementar RAM distribuída e registos de deslocamento de até 32 bits. Cada LUT possui duas saídas, pois além de poder implementar uma função lógica de 6 entradas, pode também implementar duas funções lógicas de 5 entradas (desde que as entradas sejam comuns às duas funções). Para funções com mais entradas são usadas várias LUTs. Ao permitirem implementar funções lógicas com muitas entradas, estas LUTs melhoram o desempenho do sistema, pois deixa de haver tanta necessidade de recorrer ao encadeamento de LUTs, para implementar funções lógicas complexas. Além disso aumentam significativamente a capacidade de memória distribuída, pois cada LUT, dos SLICEM, pode armazenar 64 bits. Os tempos de propagação nas LUT são independentes da função implementada e do número de funções implementadas por cada LUT (quer seja uma função de 6 entradas ou duas de 5 entradas) [Xil09c].

O bloco CMT (*Clocking & Management Tile*) é importante para garantir a qualidade do sinal de relógio, quando este faz o *drive* de diversos circuitos. Cada bloco CMT possui dois blocos DCM (*Digital Clock Manager*) e um bloco PLL (*Phase-Locked Loop*). Estes blocos têm ligações dedicadas para facilitar o uso em conjunto, mas podem ser usados separadamente. O bloco PLL permite sintetizar frequências numa gama alargada e pode ser usado para reduzir o *jitter* de sinais de relógio internos ou externos, quando usado em conjunto com o DCM. Cada bloco DCM é composto por 3 blocos funcionais: um *Delay-Locked Loop* (DLL) que é

usado para compensar o atraso do sinal de relógio, um *Digital Frequency Synthesizer* (DFS), utilizado para criar uma nova frequência de relógio a partir da frequência de entrada e um *Phase Shift* (PS) que em conjunto com o DLL, permite desfasar o sinal de relógio [Xil09c].

Uma das funcionalidades mais importantes do DCM é a eliminação do atraso do sinal de relógio (*Clock Skew*), que aparece naturalmente ao longo do sistema quando é necessário efectuar o *drive* de uma grande quantidade de *flip-flops*. A eliminação do *Clock Skew* é conseguida através de uma malha de realimentação, que elimina o atraso do sinal de relógio de forma dinâmica e independentemente da temperatura ou outros factores.

Os *transceivers RocketIO* são usados em aplicações onde os dados são recebidos série, a ritmos da ordem dos Gigabits por segundo (Gb/s). Permitem converter os dados em série para paralelo, de modo a que o processamento no FPGA possa ser efectuado a frequências relativamente baixas. Após o processamento, os *transceivers* fazem a operação inversa, ou seja, convertem os dados em paralelo para série. Os dispositivos *RocketIO* GTP presentes nos FPGAs *Virtex-5* têm uma grande flexibilidade de configuração e podem ser usados em inúmeras aplicações [Xil09b].

3.2 Circuitos sequenciais

Ao projectar circuitos sequenciais síncronos, é necessário ter em conta alguns aspectos. Qualquer circuito digital pode ser dividido em duas partes, a parte combinatória e a parte sequencial. A parte combinatória é formada pelos elementos que não possuem memória, como por exemplo os negadores, portas AND ou portas OR. Num circuito puramente combinatório a saída é determinada em função das entradas e não depende de entradas anteriores. Por outro lado, nos circuitos sequenciais a saída depende das entradas e do estado actual (que acumula a informação sobre todas as entradas anteriores), isto significa que o circuito possui memória. Os elementos básicos de memórias são as *latches* e os *flip-flops* (ou registos). Num circuito sequencial existe um sinal de relógio que define o momento em que os estados são actualizados. Antes da transição activa do relógio é necessário garantir que as entradas estão estáveis durante um intervalo de tempo mínimo (t_{SU}). Este intervalo de tempo, o tempo de *hold* (t_{hold}) e os tempos de propagação nos circuitos combinatórios vão influenciar a frequência máxima do sinal de relógio. O significado destes intervalos de tempo está esquematizado na figura 3.2.

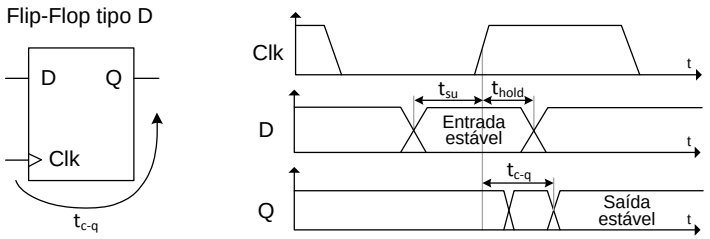


Figura 3.2: Parâmetros temporais associados a um *flip-flop* tipo D

Em sistemas digitais reconfiguráveis, os *flip-flops* e *latches* mais comuns são do tipo D. O comportamento dos *flip-flops* tipo D é muito simples: quando ocorre a transição activa do sinal de relógio o valor da entrada passa para a saída. Durante o resto do tempo, a saída mantém-se inalterada. Isto significa que os *flip-flops* são sensíveis à transição de relógio. Por sua vez, as *latches* são sensíveis ao nível do relógio, que define se a *latch* está em retenção ou transparente. Na prática é conveniente não usar *latches*, pois nem sempre o seu comportamento pode ser previsto e as simulações podem não corresponder ao comportamento real. Assim, é boa

prática garantir que o código VHDL não origina *latches*. Na figura 3.2 estão representados num diagrama temporal os diversos intervalos de tempo associados a um *flip-flop* tipo D. O significado desses intervalos é o seguinte [Mar07]:

- **Tempo de setup,** t_{SU} - Tempo mínimo em que D deve estar válido antes da transição activa do sinal de relógio (Clk);
- **Tempo de hold,** t_{hold} - Tempo mínimo em que D deve permanecer válido depois da transição activa de Clk;
- **Tempo de propagação,** t_{c-q} - Tempo de propagação para Q relativo à transição activa de Clk;

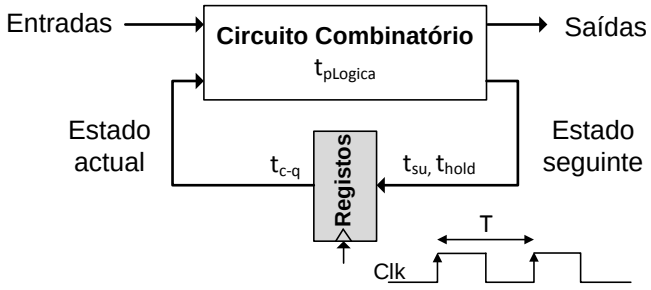


Figura 3.3: Modelo de um circuito sequencial

A figura 3.3 esquematiza a divisão entre parte combinatória e sequencial. O período mínimo T do sinal de relógio é limitado pelos parâmetros temporais do circuito e deve obedecer à relação $T \geq t_{c-q} + t_{pLogica} + t_{su}$. Quando vários *flip-flops* estão ligados em cadeia, é necessário assegurar que o t_{c-q} do primeiro *flip-flop* é maior que o t_{hold} do segundo *flip-flop*, caso contrário o segundo *flip-flop* não irá receber os dados correctamente. A relação entre t_{c-q} e t_{hold} é normalmente garantida se os *flip-flops* forem do mesmo tipo.

Em VHDL a distinção entre circuito combinatório e sequencial é feita usando dois processos. No processo sequencial, a lista de sensibilidade é composta apenas pelo sinal de relógio (CLK) e, em geral, pelo sinal de *reset* (RST). Normalmente o sinal de reset é assíncrono e as transições ocorrem no flanco ascendente do relógio, como descrito do lado esquerdo da figura 3.4. Neste processo, além da inicialização dos registos, são definidas as transições que irão ocorrer no próximo flanco ascendente do sinal de relógio. No caso de *flip-flops* tipo D, corresponde apenas a indicar que a saída do registo será igual à entrada. As entradas dos registos são atribuídas no processo combinatório.

```
sequencial:Process(CLK,RST)
Begin
   If(RST='1') Then
      --Inicialização dos registos
   Elsif rising_edge(CLK) Then
      --Transições dos registos
   End If;
End Process sequencial;
```

```
comb:Process(entradas,sinais internos)
Begin
   --Definição do próximo estado
   --dos registos

   --Definição das saídas

End Process comb;
```

Figura 3.4: Estrutura dos processos sequenciais e combinatórios em VHDL

No processo combinatório, a lista de sensibilidade é composta pelos sinais de entrada e pelos sinais internos que necessitam de ser "medidos", ou seja, os sinais que dentro do processo

são usados para fazer comparações ou que aparecem do lado direito de uma atribuição. A ferramenta de síntese utilizada (ISE) não permite que a lista de sensibilidade esteja incompleta [Xil09d], incluindo automaticamente os sinais em falta. No entanto, a ferramenta de simulação (*ModelSim*) permite listas de sensibilidade incompletas. Para que a simulação corresponda ao comportamento real, é necessário garantir que não faltam sinais na lista de sensibilidade. As operações efectuadas no processo combinatório são sobretudo atribuições aos sinais de entrada dos registos, que na próxima transição ascendente do relógio serão actualizados. São ainda definidas as saídas do circuito como indicado do lado direito da figura 3.4.

3.3 Diagrama de blocos do receptor OTU1

Nesta tese pretende-se implementar um terminal de recepção para redes ópticas de transporte (OTN). A norma G.709 define os ritmos de transmissão e o formato da trama OTN. Ao contrário de outros protocolos (SONET/SDH), a trama OTN tem a mesma organização de bytes para qualquer ritmo de transmissão. A diferença entre os vários ritmos de transmissão está na duração de cada bit. Isto significa que, em teoria, os circuitos para processar tramas OTU1, OTU2 e OTU3 apenas diferem na frequência de operação. Na prática, as implementações terão de ser adaptadas a cada caso, para cumprir as limitações temporais do *hardware*.

O receptor em estudo é projectado para receber tramas OTU1 ($2,666Gb/s$). A esta frequência é impossível processar os bits recebidos em série. No entanto, se a frequência de operação for cerca de uma ordem de grandeza inferior, já é possível utilizar um FPGA para implementar o receptor. A solução é utilizar um *transceiver RocketIO*, que permite converter os dados para paralelo. A implementação deste bloco não faz parte deste projecto, como tal assume-se que os dados de entrada já estão em paralelo. Neste projecto o barramento de dados é de 16 bits, reduzindo a frequência interna de operação em 16 vezes, para aproximadamente 167MHz.

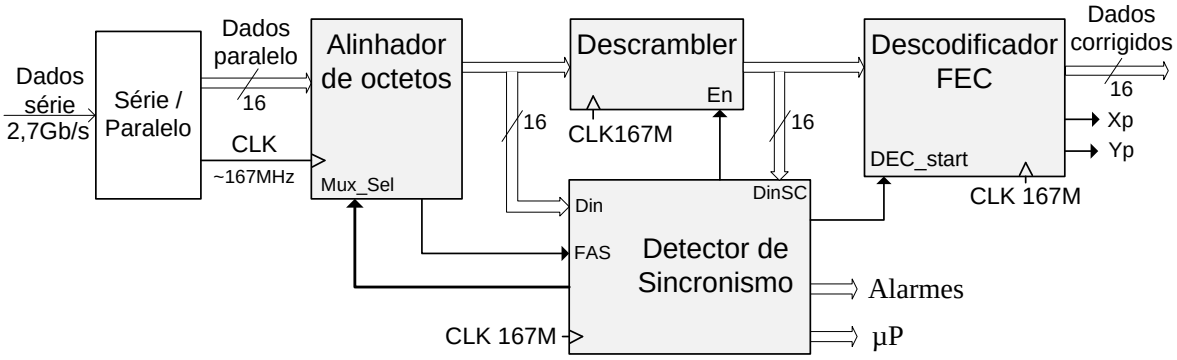


Figura 3.5: Diagrama de blocos do receptor para OTU1

Na figura 3.5 é apresentado o diagrama de blocos geral do receptor de tramas OTU1. Depois da conversão dos dados de entrada para paralelo, o bloco *Alinhador de Octetos* em conjunto com o bloco *Detector de Sincronismo* garantem o alinhamento correcto dos dados. Após esta fase, o *Descrambler* entra em acção para que os dados originais sejam recuperados. O *Detector de Sincronismo* recebe os dados descodificados pelo *Descrambler* e monitoriza os dados recebidos, analisando os cabeçalhos da trama. Este bloco comunica com o exterior através de um microprocessador (μP), indicando a existência de erros na transmissão ou alarmes na trama. Além disso controla o bloco *Descodificador FEC* através do sinal *DEC_start*. O *Descodificador FEC* corrige possíveis erros da trama, e disponibiliza os dados para o exterior,

onde a carga paga pode ser acedida. São também gerados os sinais *Xp* e *Yp* que são as coordenadas da trama. O *Yp* corresponde ao número da linha, enquanto *Xp* corresponde ao número da coluna, onde cada coluna tem o comprimento de dois bytes.

3.4 Bloco Alinhador de Octetos

Para um correcto funcionamento de todos os blocos posteriores, é necessário garantir que os octetos estejam alinhados correctamente, ou seja, o byte mais significativo do barramento de dados tem de corresponder a um byte ímpar da trama [1]. Este bloco tem ainda a função de sinalizar o início da trama.

O bloco *Alinhador de Octetos* possui como entradas o barramento de dados de 16 bits (*Din*) e 4 bits de controlo de um multiplexer interno (*MUX_SEL*), além do sinal de relógio a $167MHz$ (CLK) e o sinal de *reset* (*AClear*). As saídas são *Dout*, para os dados já alinhados, e o sinal *FAS*, que estará a '1' quando for detectada a sequência FAS (*Frame Alignment Signal*). A interface do bloco alinhador encontra-se na figura 3.6.

Figura 3.6: Interface do bloco *Alinhador de Octetos*

Uma forma simples de verificar se o alinhamento está correcto é procurar por um padrão que exista numa posição conhecida de todas as tramas. Esse padrão é o FAS, que é composto por 6 bytes, como representado na figura 3.7 onde OA1="1111 0110" e OA2="0010 1000". O FAS corresponde à sequência hexadecimal 0xF6F6F6282828, que são os primeiros 6 bytes de cada trama. O FAS deve ser verificado e detectado sempre na mesma posição, para garantir que se trata efectivamente da sequência inicial da trama e não de um conjunto de bits que pode aparecer noutras posições aleatórias da trama.

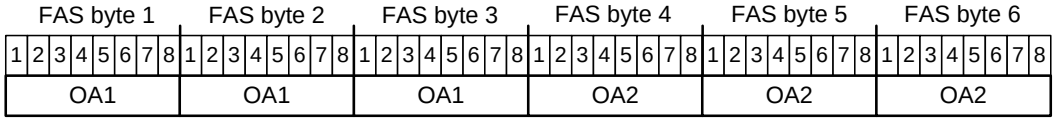


Figura 3.7: Sequência de bits do *Frame Alignment Signal* (FAS)

A abordagem escolhida baseia-se na utilização de um multiplexer 16:1 de 48 bits e um comparador de 48 bits. O esquema do circuito *Alinhador de Octetos* encontra-se na figura 3.8 onde os registos 1 a 4 servem para armazenar um total de 63 bits de entrada consecutivos (*MUX_IN*), que vão sendo deslocados 16 posições por cada ciclo de relógio. Os 16 bits menos significativos de *MUX_IN* são sempre a palavra de entrada mais recente.

A dimensão de *MUX_IN* é justificada tendo em conta que o FAS tem 48 bits e que este pode começar em qualquer uma das 16 posições do barramento de entrada. Assim, com um desfasamento zero, o FAS aparecerá nos 48 bits menos significativos de *MUX_IN*. No outro extremo, a sequência aparecerá deslocada 15 bits, ou seja, em *MUX_IN[62 . . . 15]*. O registo 5 (48 bits) guarda a saída do multiplexer e está ligado directamente ao comparador, que verifica se o registo possui a sequência pretendida. A detecção do padrão é sinalizada colocando a

[1] O primeiro byte da trama tem o número 1.

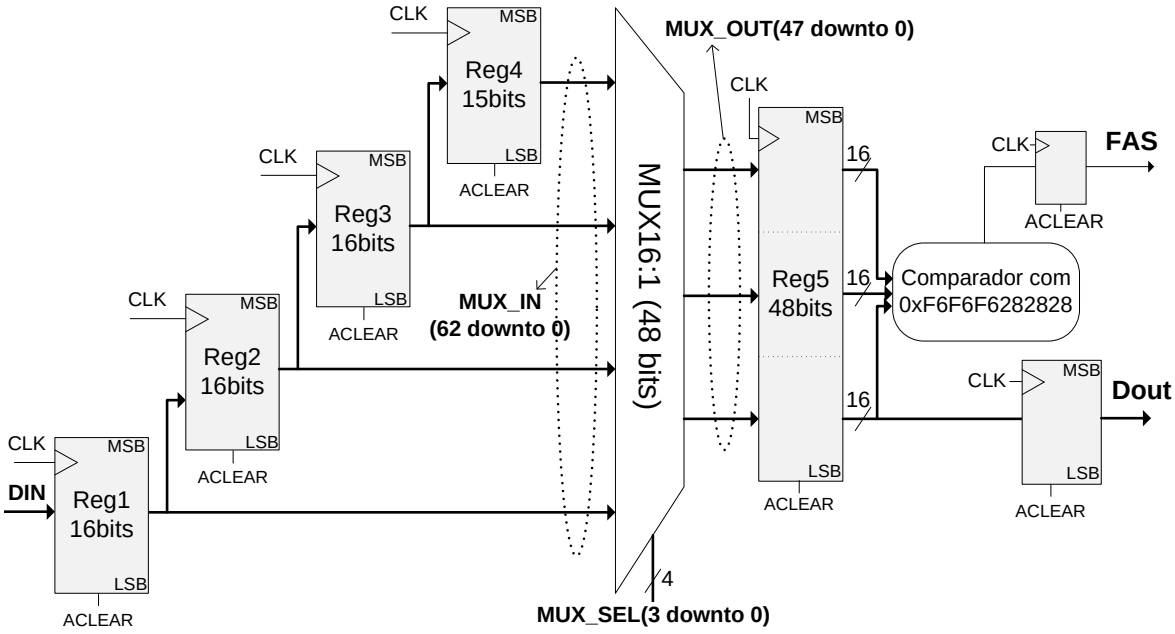


Figura 3.8: Circuito interno do *Alinhador de Octetos*

saída *FAS* no nível lógico '1'. O registo entre o multiplexer e o comparador permite que a frequência de operação seja maior, pois o atraso do circuito combinatório é dividido em duas etapas.

O multiplexer de 48 bits da figura 3.8, pode ser visto como um conjunto de 48 multiplexers 16:1. Considerando que o bit mais significativo de *MUX_IN* tem o número 62 e o menos significativo tem o número 0, pode-se dizer que um MUX 16:1 corresponde aos bits *MUX_IN[15 . . . 0]*, outro corresponde aos bits *MUX_IN[16 . . . 1]* e assim sucessivamente até *MUX_IN[62 . . . 47]*. Todos os MUXs são controlados pelos mesmos 4 bits de selecção, deste modo com *MUX_SEL* = "0000", *MUX_OUT* corresponde aos 48 bits menos significativos de *MUX_IN* e com *MUX_SEL* ="1111", tem-se em *MUX_OUT* os 48 bits mais significativos da entrada do multiplexer.

O controlo dos bits de selecção do multiplexer é efectuado pelo bloco *Detector de Sincronismo*, que irá sucessivamente incrementar o sinal *MUX_SEL* até que seja encontrado o FAS. O espaçamento temporal entre cada incremento é de duas tramas, deste modo garante-se que toda a trama é analisada com o mesmo alinhamento. Quando o FAS for encontrado, o sinal *MUX_SEL* deve ser mantido indefinidamente até que o FAS deixe de ser detectado. A saída *Dout*, corresponde aos 16 bits menos significativos do registo 5. Deste modo quando *FAS* transita de '0' para '1', a saída de dados (*Dout*) transita de 0xF628 para 0x2828.

3.4.1 Síntese do Alinhador de Octetos

A tabela 3.1 mostra os recursos usados pelo bloco *Alinhador de Octetos* após síntese e implementação com a ferramenta ISE. É também indicada a frequência máxima aproximada que o circuito suporta.

O valor da frequência máxima foi obtido definindo no ISE uma limitação temporal (*timing constraint*) de $200MHz$, que é superior ao necessário ($167MHz$) para garantir uma margem de segurança. A frequência máxima é bastante superior ao pretendido, portanto este bloco cumpre os requisitos temporais. Como era de esperar a área ocupada não é muito significativa. É possível verificar que o número de *flip-flops* utilizados pelo ISE corresponde ao número de

Alinhador de Octetos	
Número de *flip-flops*	128 (<1%)
Número de LUTs	113 (<1%)
Frequência máxima	$399MHz$

Tabela 3.1: Resultados da síntese do *Alinhador de Octetos* para o FPGA LX330T

flip-flops representados na figura 3.8.

3.5 *Scrambler/Descrambler*

O funcionamento do *Scrambler* descrito na recomendação G.709, baseia-se em gerar uma sequência de bits pseudo-aleatória, utilizando um LFSR. A sequência de bits é depois somada em módulo 2 (XOR) aos bits de dados, passando o resultado a ter também características aleatórias. A sequência gerada por um LFSR repete-se ao longo do tempo e a frequência de repetição depende da ordem do polinómio gerador. No caso da norma OTN o polinómio gerador tem grau 16, portanto a sequência repete-se a cada 65535 bits ($2^{16}-1$). A utilização do *Scrambler* garante que a trama a transmitir não possui longas sequências de '0's ou '1's, facilitando a extracção do sinal de relógio no receptor. Torna também menos provável a repetição da sequência de alinhamento ao longo da trama. O bloco de *Descrambler*, usado no receptor para recuperar a informação original, é igual ao bloco *Scrambler*, pois $((S \bigoplus D) \bigoplus S) = D$, onde D representa os dados e S a sequência do *Scrambler*.

A sequência pseudo-aleatória é gerada pelo polinómio $1 + x + x^3 + x^{12} + x^{16}$, que pode ser implementado pelo LFSR da figura 3.9, tal como definido na norma G.709. No entanto, este circuito não pode ser aplicado directamente neste projecto, porque no circuito da figura 3.9 os dados de entrada são recebidos em série. Assim é necessário criar um circuito que produza resultados coerentes com a norma, mas que funcione com um barramento de dados de 16 bits.

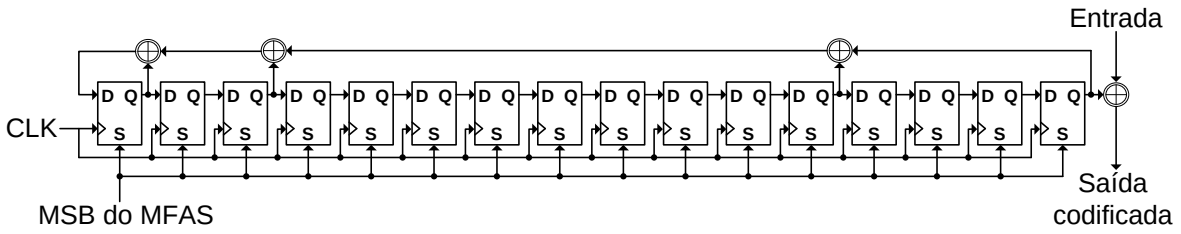


Figura 3.9: *Scrambler/Descrambler* para dados em série, baseado em LFSR [IT03b]

Uma possível implementação é descrita em [PM07] e baseia-se em guardar em memória toda a sequência pseudo-aleatória. Esta solução requer bastantes recursos, mas minimiza o atraso do circuito combinatório. Neste projecto uma solução baseada em LFSR é mais indicada pois ocupa poucos recursos e o atraso não é significativo.

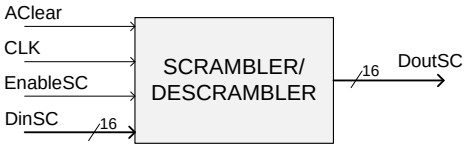


Figura 3.10: Interface do *Scrambler/Descrambler* implementado

A interface do bloco a implementar apresentada na figura 3.10 é composta pela entrada de dados *DinSC* e a respectiva saída codificada (*DoutSC*). Possui também uma entrada de *enable* (*EnableSC*) que permite seleccionar o modo de funcionamento. Com a entrada *EnableSC* a '0', o circuito está em modo transparente, ou seja, a saída será igual à entrada, apenas atrasada um ciclo de relógio. Este modo de funcionamento é útil uma vez que nem todos os bytes da trama são codificados. Com *EnableSC* a '1', o circuito codifica os bits, de acordo com a norma.

O *Scrambler* é inicializado quando o sinal de *EnableSC* transita de '0' para '1'. O processo de inicialização corresponde a colocar todos os *flip-flops* do LFSR a '1', o que é muito importante pois as operações de *Scrambler* e *Descrambler* só são complementares se ambos os blocos forem inicializados com o mesmo valor e no mesmo ponto da trama. O *Scrambler/Descrambler* é inicializado logo após o aparecimento da sequência de alinhamento FAS, ou seja, de cada vez que é recebido o bit mais significativo (MSB - *Most Significant Bit*) do campo MFAS.

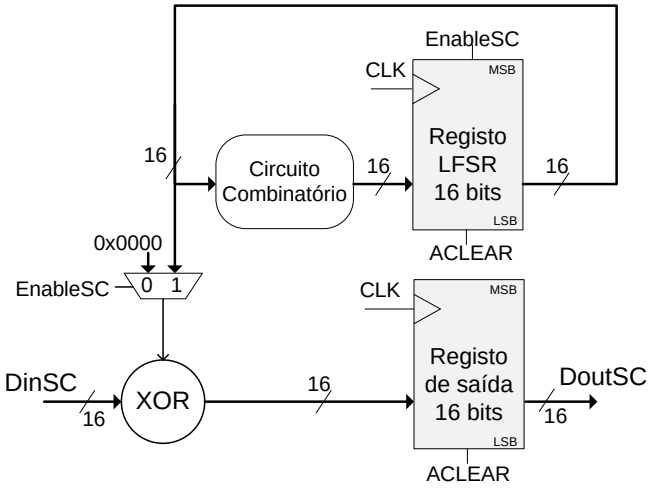


Figura 3.11: Diagrama de blocos do *Scrambler* para dados em paralelo [Saw02]

O circuito da figura 3.11 esquematiza a arquitectura do *Scrambler/Descrambler* utilizada neste projecto. Esta implementação baseia-se no circuito descrito em [Saw02]. É essencialmente composta por dois registos de 16bits, um XOR de 16 bits e um circuito combinatório formado por uma cadeia de XORs. Esta solução utiliza um registo (LFSR) que guarda 16 bits da sequência pseudo-aleatória. O circuito combinatório é responsável por calcular os próximos 16 bits da sequência. Ao mesmo tempo é efectuada a operação XOR entre a entrada e a sequência de 16 bits actual, cujo resultado é guardado no registo de saída.

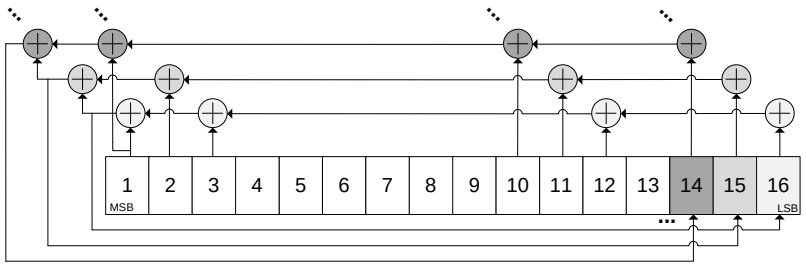


Figura 3.12: Circuito combinatório do *Scrambler*

O circuito combinatório da figura 3.11 é esquematizado na figura 3.12 onde cada número

representa um flip-flop do registo LFSR. A cadeia de XORs é semelhante ao circuito *Scrambler* série, mas de algum modo replicada 16 vezes. Comparando com o circuito série, pode-se observar que neste caso os bits são deslocados 15 posições para a direita. Por exemplo, o resultado do XOR dos bits 16, 12, 3 e 1 é guardado na posição 16 do registo, em vez da posição 1 como no *Scrambler* série.

Os restantes bits são calculados deslocando a cadeia de XORs sucessivamente para a esquerda, como exemplificado na figura 3.12. Na prática não é necessário projectar manualmente todo o circuito combinatório, pois a linguagem de descrição de hardware permite criá-lo facilmente utilizando a instrução "`for i in 0 to 15 generate`".

3.5.1 Síntese do Scrambler/Descrambler

A tabela 3.2 mostra os recursos usados pelo bloco *Scrambler/Descrambler* após síntese e implementação com ferramenta ISE. É também indicada a frequência máxima aproximada que o circuito suporta. Sendo um circuito bastante simples, são necessários poucos recursos. Os 32 *flip-flops* correspondem aos dois registos de 16 bits representados na figura 3.11. A lógica combinatória é algo complexa, mas após simplificação pelo ISE, são usadas apenas 48 LUTs. Os requisitos temporais são facilmente cumpridos.

Scrambler/Descrambler	
Número de *flip-flops*	32 (<1%)
Número de LUTs	48 (<1%)
Frequência máxima	$602MHz$

Tabela 3.2: Resultados da síntese do *Scrambler/Descrambler* para o FPGA LX330T

3.6 Detector de Sincronismo

O bloco *Detector de Sincronismo* desempenha várias funções:

- Efectuar a sincronização de trama e multi-trama;
- Gerar as coordenadas da trama;
- Controlar os blocos *Alinhador de Octetos*, *Descrambler* e *Descodificador FEC*;
- Analisar o cabeçalho, para monitorização de tráfego;

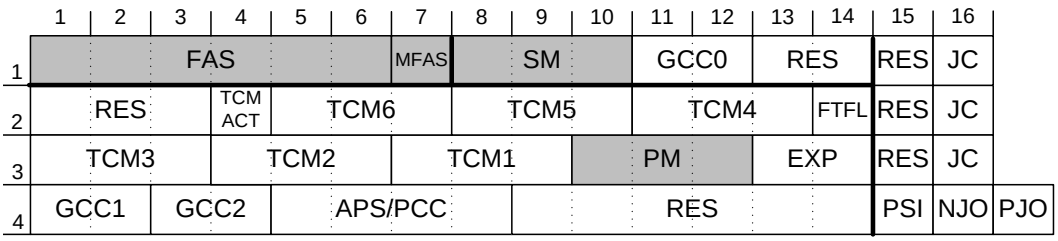


Figura 3.13: Campos importantes do cabeçalho OTN

Na figura 3.13 estão realçados a cinzento os campos do cabeçalho usados pelo *Detector de Sincronismo*, que são essenciais em equipamentos de redes OTN. Estes campos permitem

implementar as funções listadas anteriormente. Os campos restantes são sobretudo partes não definidas (RES e EXP), canais de comunicação genéricos (GCC) e os campos TCM, usados apenas para ligações em cascata. Os campos JC, PSI, NJO e PJO do cabeçalho OPU são importantes para a desmultiplexagem da carga-paga, que não é o objectivo deste projecto. Assim restam os campos FAS e MFAS, usados para sincronismo de trama e multi-trama e os campos SM e PM, usados para monitorização do tráfego.

3.6.1 Sincronismo de trama e multi-trama

Existem dois tipos de alinhamento para garantir um sincronismo correcto. O alinhamento de trama está relacionado com o campo FAS e serve para delimitar o início e fim de trama. O alinhamento de multi-trama está relacionado com o contador MFAS, que permite utilizar estruturas de dados que se espalham por várias tramas.

No alinhamento de trama são usados dois sinais, o OOF (*Out of frame*) e o LOF (*Loss of frame*). O sinal OOF é colocado a '1', sempre que a sequência FAS não é detectada correctamente. Para evitar que falhas de sincronismo ocorram devido a erros pontuais, existe o LOF que apenas é declarado após a recepção de um certo número de tramas erradas consecutivas. Verificando o estado do sinal LOF, é possível saber se a informação recebida da trama é válida. O processo de alinhamento de multi-trama é muito semelhante ao anterior, mas usando o byte MFAS. Neste caso os sinais definidos são o OOM (*Out of Multiframe*) e o LOM (*Loss of Multiframe*).

O comportamento dos sinais OOF, LOF, OOM e LOM é descrito na norma G.798 [IT06]. A atribuição dos sinais é definida recorrendo a máquinas de estados, que são apresentadas de seguida.

Detecção de Out of frame - OOF

A norma G.798 define dois estados para validação do alinhamento de trama, o *Out-Of-Frame* (OOF) e o *In-Frame* (IF). A máquina de estados deve transitar para IF quando o FAS é detectado em duas tramas consecutivas. A transição de IF para OOF deve acontecer quando o FAS não for detectado correctamente durante 5 tramas consecutivas [IT06].

Na prática são usados dois estados intermédios, o ST_TO_IF e o ST_TO_OOF. A máquina transita para ST_TO_IF quando está no estado ST_OOF e a entrada FAS e colocada a '1' [2]. Se o FAS for confirmado na próxima trama, a máquina transita para o estado ST_IF. Quando a detecção do FAS falha, a máquina passa para ST_TO_OOF. Neste estado existe um contador de tramas consecutivas onde o FAS não é detectado (*TramasNaoOk*). Se FAS≠1 e se foram detectadas 5 tramas consecutivas erradas, então a máquina passa para ST_OOF. Se FAS=1, então o estado passa a ser ST_IF. Na figura 3.14 está esquematizada a máquina de estados para a detecção de OOF.

A máquina de estados da figura 3.14 também é responsável por definir e actualizar as coordenadas da trama. No estado ST_OOF as coordenadas são incrementadas mesmo que ainda não tenha sido detectado qualquer FAS. Assim que o FAS é detectado, as coordenadas são inicializadas no valor correcto, ou seja, Xp=3 e Yp=0. A coordenada inicial Xp=3 corresponde aos bytes 7 e 8 da trama [3], já que são recebidos dois bytes de cada vez e os primeiros dois bytes recebidos correspondem à coordenada Xp=0. A coordenada Yp corresponde ao número da linha, ou seja, Yp=0 corresponde à primeira linha e Yp=3 à quarta linha. Para armazenar Yp é necessário um registo de dois bits. Já Xp varia de 0 a 2039, onde a coordenada Xp=2039 corresponde aos bytes 4079 e 4080 da trama, portanto são necessários 11 bits. As coordenadas

[2] O sinal FAS é colocado a '1' pelo bloco *Alinhador*, sempre que a sequência 0xF6F6F6282828 é encontrada.
[3] O primeiro byte da trama tem número 1.

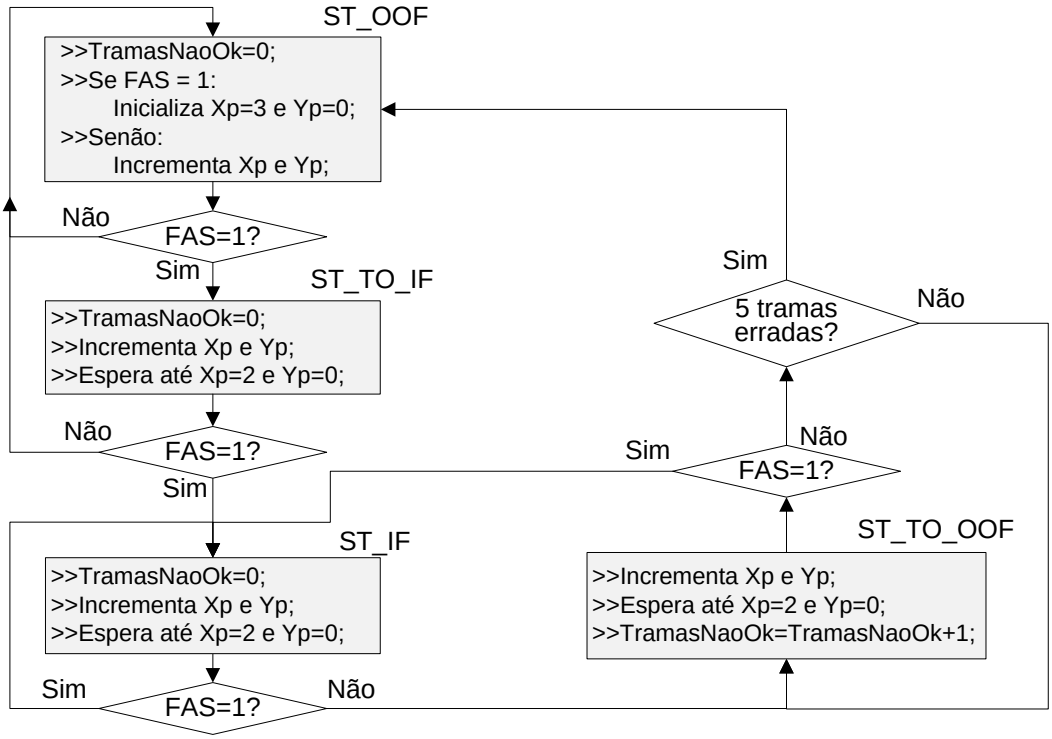


Figura 3.14: Máquina de estados para a detecção de OOF

são incrementadas em qualquer dos estados, mas antes de incrementar é necessário verificar se foi atingido o limite. Se Xp=2039 significa que se atingiu o fim de uma linha portanto o próximo Xp será zero.

Como no estado ST_OOF a posição do FAS ainda não é conhecida, é necessário verificar o FAS em cada ciclo de relógio. Nos restantes estados, o FAS apenas é verificado uma vez por trama, imediatamente após a recepção dos primeiros 6 bytes, ou seja, com Xp=2 e Yp=0. No resto do tempo as coordenadas são actualizadas, mas os estados e contadores são mantidos.

O sinal OOF é colocado a '1' quando a máquina de estados está em ST_OOF ou ST_TO_IF. Quando a máquina está nos estados ST_IF ou ST_TO_OOF, o sinal OOF é colocado a '0', indicando que o sincronismo da trama está correcto.

Detecção de Loss of frame - LOF

O sinal *Loss of frame* (LOF) deve ser declarado (LOF='1'), quando o processo de alinhamento de trama está OOF durante $3ms$. No caso de OOF estar intermitente, o temporizador não deve ser reiniciado até que o estado IF persista continuamente durante $3ms$. O sinal LOF deve ser limpo (LOF='0') quando o estado IF persiste continuamente durante $3ms$ [IT06].

Uma forma simples de implementar o temporizador de $3ms$, é usar um contador que é incrementado por cada trama recebida. No caso do OTU1 cada trama tem a duração de $48{,}971\mu s$, portanto $3ms$ equivale a receber 62 tramas. Serão necessários dois contadores, um para contar o tempo que o sistema está ST_OOF ou ST_TO_IF [4] (*TramasNaoOk*) e outro para contar o tempo que o sistema está ST_IF ou ST_TO_OOF [5] (*TramasOk*). A máquina de estados para detecção de LOF é apresentada na figura 3.15 e contém 3 estados: ST_LOF, ST_NORMAL e ST_TO_LOF. De acordo com a norma, a máquina apenas transita para o

[4] Os estados ST_OOF e ST_TO_IF correspondem ao estado OOF referido na norma.

[5] Os estados ST_IF e ST_TO_OOF correspondem ao estado IF referido na norma.

estado ST_NORMAL, quando o processo de alinhamento de trama se encontra em ST_IF ou ST_TO_OOF durante $3ms$ consecutivos. A transição para o estado intermédio ST_TO_LOF acontece, quando a máquina está no estado ST_NORMAL e o processo de alinhamento passa para ST_OOF. No estado ST_TO_LOF é utilizado o contador *TramasOk* para verificar se o sistema está ST_IF ou ST_TO_OOF durante $3ms$ consecutivos. Caso se verifique esta situação, o sistema passa para ST_NORMAL e o contador *TramasNaoOk* é reiniciado. Se ST_OOF estiver intermitente, o contador *TramasNaoOk* não é reiniciado e se atingir $3ms$ (62 tramas), a máquina passa para o estado ST_LOF.

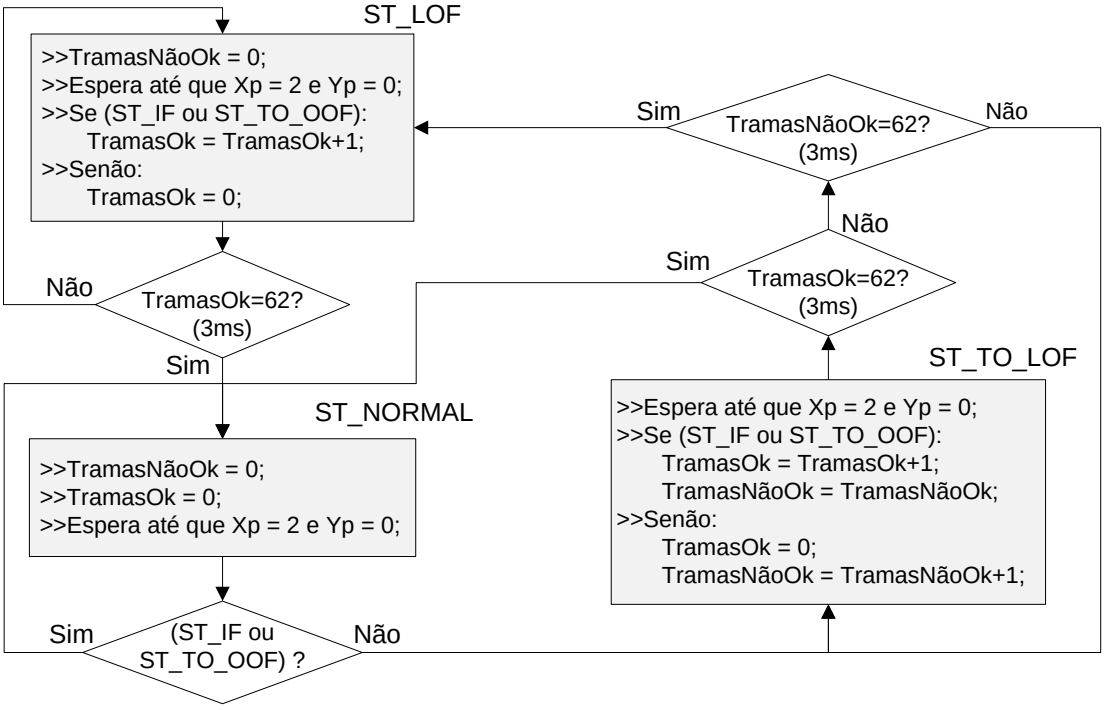


Figura 3.15: Máquina de estados para a detecção de LOF

O sinal LOF é definido externamente à máquina de detecção de LOF, mas depende do seu estado. Assim, se a máquina de estados estiver no estado ST_LOF, o sinal LOF é declarado (LOF=‘1’), caso contrário o sinal LOF é limpo (LOF=‘0’).

Detecção de Out of Multiframe - OOM

O alinhamento de multi-trama é baseado no byte MFAS. O byte MFAS é um contador de 8 bits que é incrementado em cada trama. A norma define dois estados para o processo de validação do MFAS, o *Out-Of-Multiframe* (OOM) e o *In-Multiframe* (IM). A máquina de estados deve transitar de IM para OOM quando o MFAS recebido não coincide com o número esperado, em 5 tramas consecutivas. No estado OOM, o alinhamento de multi-trama deve ser recuperado e o contador de multi-tramas deve tomar o valor do último MFAS recebido. A transição para o estado IM ocorre quando uma sequência MFAS válida é encontrada em duas tramas consecutivas. A sequência MFAS é considerada válida, se o MFAS da segunda trama é um incremento da primeira [IT06].

À semelhança do que acontece na máquina de estados para detecção de OOF, são usados os estados intermédios ST_TO_OOM e ST_TO_IM. Quando a máquina de estados está em ST_OOM e o MFAS recebido está correcto a máquina passa para o estado ST_TO_IM, tal como ilustrado na figura 3.16. Se na trama seguinte o MFAS estiver novamente correcto,

ocorre a transição para ST_IM. Se for detectado um valor de MFAS incorrecto, o sistema passa para o estado ST_TO_OOM. Neste estado existe um contador de tramas para verificar se o MFAS está incorrecto durante 5 tramas consecutivas, se isto acontecer o sistema entra no estado ST_OOM. No estado ST_OOM o registo interno MFAS continua a ser incrementado. Só no estado ST_TO_IM o valor MFAS é actualizado para o valor recebido.

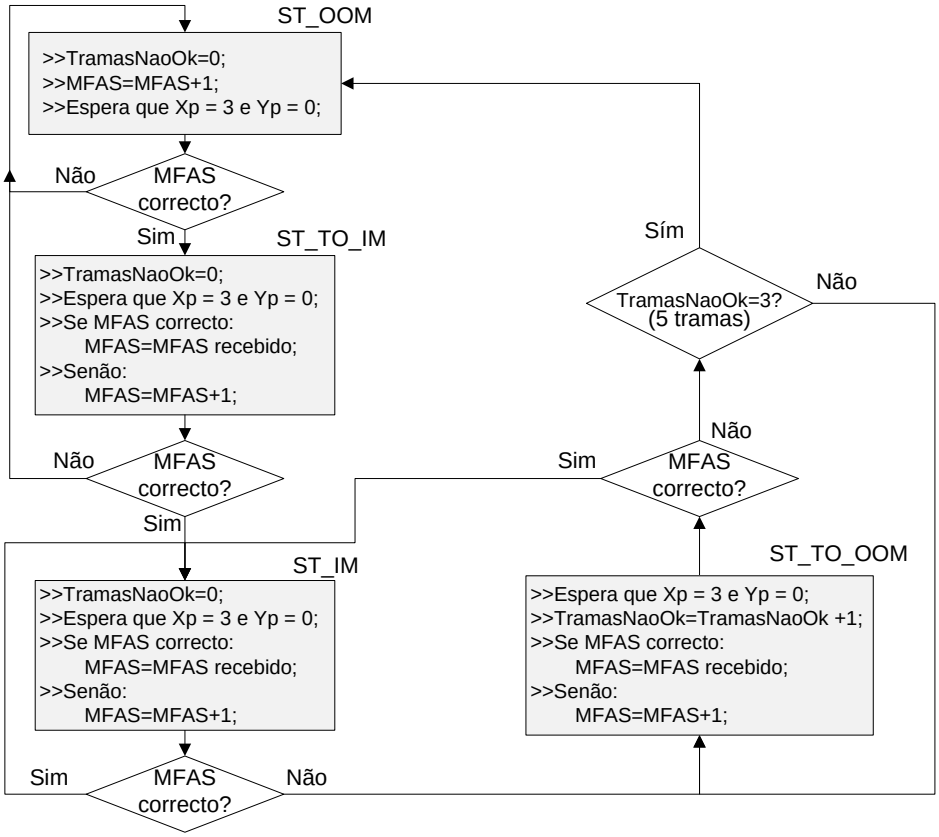


Figura 3.16: Máquina de estados para a detecção de OOM

O sinal de saída OOM, é colocado a '1' quando a máquina de estados de detecção de OOM está nos estados ST_OOM ou ST_TO_IM. Quando a máquina está nos estados ST_IM ou ST_TO_OOM, a saída OOF é colocada a '0', indicando que o sincronismo de multi-trama está correcto.

Detecção de Loss of Multiframe - LOM

O sinal *Loss of Multiframe* (LOM) deve ser declarado (LOM='1'), quando o processo de alinhamento de multi-trama está no estado ST_OOM durante $3ms$. O sinal LOM deve ser imediatamente limpo (LOM='0'), assim que a máquina entra no estado ST_IM [IT06].

De modo semelhante à máquina de estados para detecção de LOF, a máquina de estados da figura 3.17 usa uma variável para contar tramas, que funciona como um temporizador. No entanto, neste caso apenas é necessário contar o número de tramas consecutivas em que o sistema está em ST_OOM ou ST_TO_IM, visto que se ocorrer o estado ST_IM ou ST_TO_OOM, a transição para ST_NORMAL é imediata.

Mais uma vez o sinal de saída LOM é definido externamente à máquina de estados. Se a máquina estiver no estado ST_LOM, o sinal LOM é declarado (LOM='1'), caso contrário o sinal LOM é limpo (LOM='0').

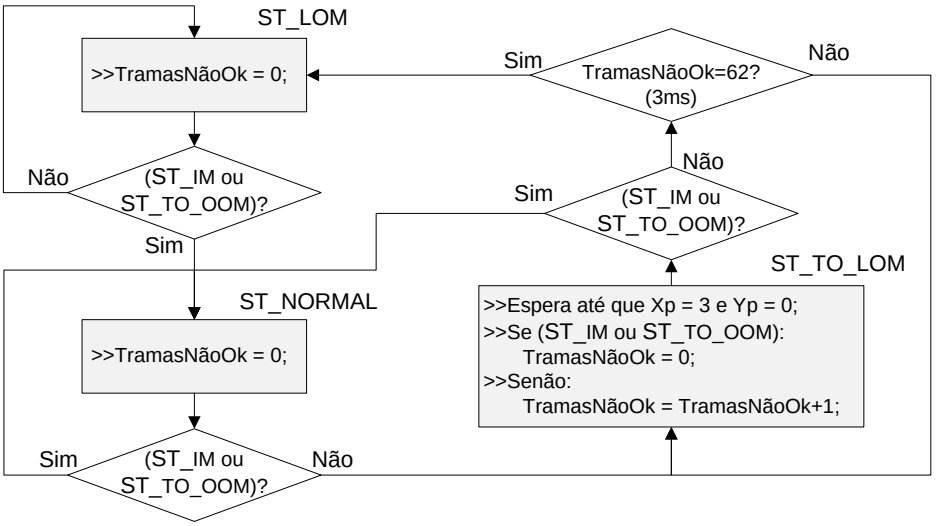


Figura 3.17: Máquina de estados para a detecção de LOM

3.6.2 Controlo do *Descrambler* e Alinhador de Octetos

Após as coordenadas da trama estarem correctas o controlo do *Descrambler* é bastante simples. Sabendo que no transmissor apenas o campo FAS não passou pelo *Scrambler*, é necessário activar o *Scrambler* a partir das coordenadas Xp=3 e Yp=0. Neste projecto os registos só são actualizados na transição de relógio ascendente, além disso o bloco *Descrambler* tem um atraso interno de um ciclo de relógio. Portanto a desactivação do *Descrambler* é efectuada quando Xp=2037 e Yp=3 e a activação é feita quando Xp=1 e Yp=0. O bloco *Detector de Sincronismo* necessita da saída do *Descrambler* para aceder à informação do cabeçalho.

O controlo do bloco *Alinhador de Octetos* resume-se a definir os sinais de selecção do multiplexer interno do *Alinhador*. O sinal de controlo do multiplexer (MUX_SEL) do bloco alinhador tem 4 bits, o que equivale a 16 alinhamentos diferentes correspondentes às 16 posições do barramento de dados. A função do bloco *Detector de Sincronismo* é definir o sinal MUX_SEL e verificar se o FAS é detectado. O sinal MUX_SEL é incrementado de duas em duas tramas, para garantir que pelo menos uma trama completa é verificada com o mesmo alinhamento. Quando o FAS for detectado é porque o alinhamento está correcto e MUX_SEL é mantido. O sinal MUX_SEL só volta a ser alterado se o sinal OOF for declarado, ou seja, se forem detectadas 5 tramas consecutivas com FAS errado.

3.6.3 Section Monitoring - SM (cabeçalho OTU)

O campo SM (*Section Monitoring*) ocupa as colunas 8, 9 e 10 da primeira linha da trama. A organização do campo SM está indicada na figura 3.18. Este campo possui um byte para identificação, um para detecção de erros de paridade e outro para alarmes. O primeiro byte é o TTI (*Trail Trace Identifier*) que contém informações sobre a origem (SAPI) e destino (DAPI) da trama e ainda uma parte específica para o operador. A estrutura TTI divide-se por várias tramas, no total são necessárias 64 tramas para receber toda a informação do campo TTI. O contador MFAS indica o significado do campo TTI. Quando MFAS[5,4]=00, o campo TTI tem informação relativa a SAPI, com MFAS[5,4]=01, o campo TTI tem informação relativa a DAPI e com MFAS[5]=1, TTI representa informação específica do operador. Toda a estrutura TTI é guardada em memória, para poder ser lida pelo microprocessador.

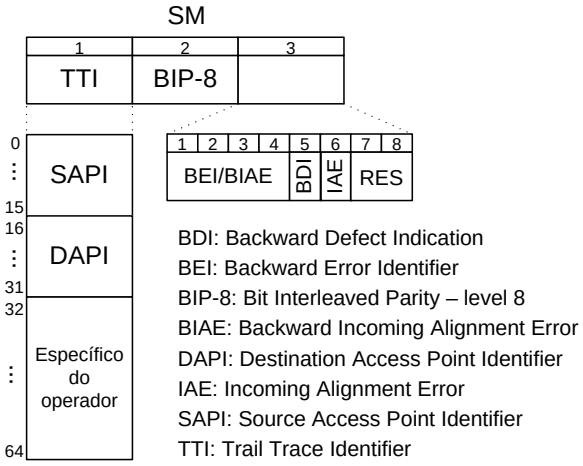


Figura 3.18: Constituição do campo SM do cabeçalho OTU

O segundo byte do campo SM é usado para detectar erros na trama, usando um código BIP-8 (*Bit Interleaved Parity-8*) com paridade par. No transmissor, o BIP-8 é calculado sobre todos os bytes do OPU (colunas 15 a 3824) da trama i e inserido no campo BIP-8 duas tramas depois (trama $i + 2$). O campo BIP-8 é calculado fazendo o XOR byte-a-byte de todos os bytes do OPU. No receptor é efectuado o mesmo cálculo e o resultado é comparado com o campo BIP-8, recebido duas tramas antes. O processo de verificação de erros de BIP-8 está representado na figura 3.19. A operação XOR é usada para comparar o valor calculado com o valor recebido. Se a comparação for diferente de zero, significa que ocorreram erros nos bytes do OPU. Este método de detecção de erros tem algumas limitações, já que apenas detecta um número ímpar de erros e não é possível saber o número total de erros ocorridos. Para contabilizar o número de erros de BIP-8 detectados, existe um registo que pode ser lido pelo microprocessador externo.

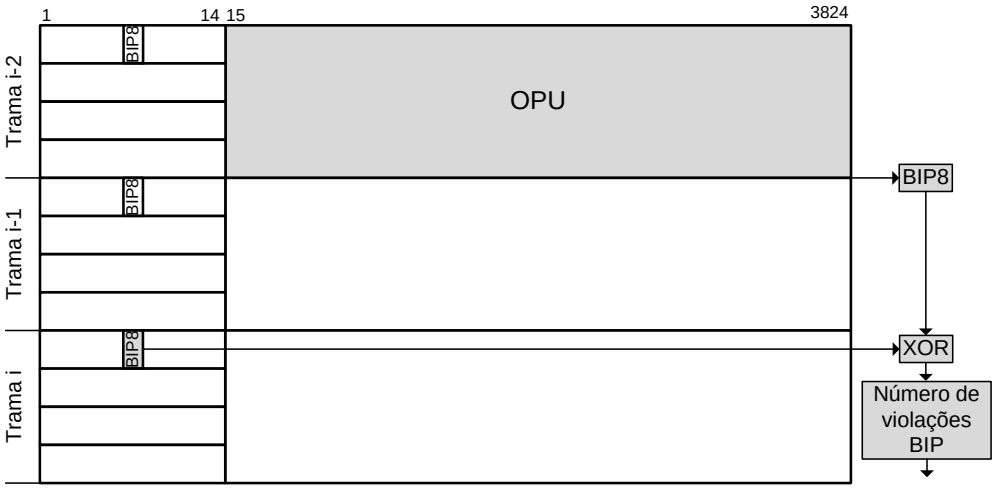


Figura 3.19: Verificação de erros de BIP-8 (SM), no receptor [IT06]

Uma ligação entre dois equipamentos terminais A e B em redes OTN, pressupõe a existência de um canal de comunicação de A para B ($A \rightarrow B$) e outro de B para A ($B \rightarrow A$). Portanto cada terminal terá de ter um equipamento de transmissão e outro de recepção. O terceiro byte do campo SM permite que cada terminal informe o outro acerca de erros ou

alarmes. O campo de 4 bits BEI (*Backward Error Indication*) do SM, é usado para indicar o número de erros de BIP-8 detectados. Se B recebe um BEI com valores entre 0000 e 1000, significa que A detectou entre 0 e 8 erros de BIP-8, que ocorreram na transmissão $B \rightarrow A$. De forma idêntica, se B detectar erros de BIP-8 na trama recebida, vai preencher campo BEI com o número de erros detectados e vai transmitir a informação para A. O significado dos bits BEI (primeiros 4 bits do terceiro byte do SM) é apresentado na tabela 3.3. De notar que nem todas as combinações são válidas e a sequência 1011 é usada para indicar a ocorrência de BIAE que se explica mais à frente.

bits BEI/BIAE (SM)	BIAE	Erros BIP-8
0000	falso	0
0001	falso	1
0010	falso	2
0011	falso	3
0100	falso	4
0101	falso	5
0110	falso	6
0111	falso	7
1000	falso	8
1001 ou 1010	falso	0
1011	verdadeiro	0
1100 até 1111	falso	0

Tabela 3.3: Significado dos bits BEI/BIAE do cabeçalho SM [IT03b]

No caso do terminal A estar a ter problemas de alinhamento, A envia a B a indicação através do sinal IAE (*Incoming Alignment Error*), presente no bit 6 do byte 3 do SM. O sinal IAE é colocado a '1' quando ocorre um erro de alinhamento, caso contrário é colocado a '0'. Para B declarar o alarme IAE=1, é necessário receber 5 tramas consecutivas com o sinal IAE=1. Por outro lado, é necessário receber 5 tramas consecutivas com IAE=0, para limpar o alarme (IAE=0). Quando B declara o alarme IAE=1, a contagem de erros de BIP-8 em B é ignorada. Neste caso o campo BEI passa a ter o significado de BIAE (*Backward Incoming Alignment Error*), sendo preenchido com o valor 1011, tal como indicado na tabela 3.3. É necessário receber 3 tramas consecutivas com BIAE=1011 para que este sinal seja validado [IT06].

Outro sinal presente no campo SM é o BDI (*Backward Defect Indication*). Quando A envia para B um sinal de manutenção, B coloca o sinal BDI=1 e envia para A. Para validação do alarme BDI, é necessário receber 5 tramas com a sinal BDI=1. Para confirmar a recepção de BDI=0 são também necessárias 5 tramas consecutivas com o bit BDI=0. Os sinais de manutenção que dão origem ao sinal BDI, são analisados mais à frente.

3.6.4 Section Monitoring - PM (cabeçalho ODU)

O campo PM (*Path Monitoring*) tem uma estrutura semelhante ao campo SM, com excepção da parte de alarmes. Outra diferença é a sua localização na trama. Enquanto o campo SM pertence ao cabeçalho OTU, PM pertence ao cabeçalho ODU. A figura 3.20 indica a constituição do campo PM. Tal como acontece no SM, os 2 primeiros bytes do PM são respectivamente o TTI e o BIP-8, cujo funcionamento já foi descrito. O funcionamento do bit BDI do PM é igual ao funcionamento do sinal BDI do SM.

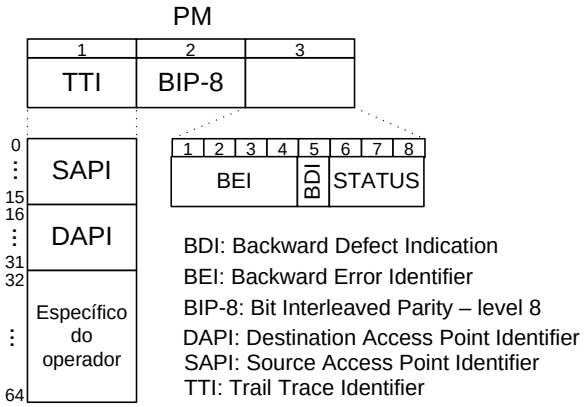


Figura 3.20: Constituição do campo PM do cabeçalho OTU

O campo BEI é usado apenas para indicar o número de erros de BIP-8 ocorridos. O BEI recebido por B indica o número de erros BIP-8 que A detectou. Por outro lado se B detectar erros de BIP-8 na trama recebida, envia para A o número de erros detectados através do campo BEI, de acordo com a tabela 3.4. De notar que existem combinações não definidas, que são consideradas 0.

bits BEI (PM)	Erros BIP-8
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001 até 1111	0

Tabela 3.4: Significado dos bits BEI do cabeçalho PM [IT03b]

No campo PM são utilizados 3 bits (STATUS) que indicam a presença de sinais de manutenção, de acordo com a tabela 3.5. A sequência 001 indica que não há sinais de manutenção. Os sinais de manutenção possíveis são LCK, OCI e AIS. Algumas sequências não são definidas e apenas podem ocorrer devido a erros.

O sinal de manutenção AIS (*Alarm Indication Signal*) consiste em colocar todos os bits do sinal ODU a '1', com excepção do byte FTFL. O sinal OCI (*Open Connection Indication*) é enviado de $A \rightarrow B$ quando A não recebe nenhum sinal vindo de B. Este sinal consiste na repetição do padrão 0110 0110 em todos os bytes do ODU. O sinal LCK (*Locked*) consiste na repetição do padrão 0101 0101 em todos os bytes do ODU e é gerado a pedido do operador, para bloquear o acesso ao utilizador. Estes sinais de manutenção não alteram o cabeçalho de alinhamento, nem o cabeçalho OTU [AGI01].

bits STATUS (PM)	Estado
000	Reservado para utilização futura
001	Sinal normal
010 até 100	Reservado para utilização futura
101	Sinal de manutenção: LCK
110	Sinal de manutenção: OCI
111	Sinal de manutenção: AIS

Tabela 3.5: Significado dos bits STATUS do cabeçalho PM [IT03b]

3.6.5 Síntese do Detector de Sincronismo

A tabela 3.6 mostra os recursos usados pelo bloco *Detector de Sincronismo*. Devido à complexidade do bloco, a frequência máxima permitida é de apenas $208MHz$, ainda assim superior ao necessário. A área ocupada é bastante superior à de outros blocos analisados anteriormente, mas ainda bastante reduzida, face à capacidade do FPGA.

Detector de Sincronismo	
Número de *flip-flops*	372 (<1%)
Número de LUTs	422 (<1%)
Frequência máxima	$208MHz$

Tabela 3.6: Resultados da síntese do *Detector de Sincronismo* para o FPGA LX330T

3.7 Descodificador FEC

Na trama OTN, além do cabeçalho e da carga paga, existe um conjunto de bytes (*Forward Error Correction - FEC*) que possibilitam a correcção de possíveis erros da trama. Os bytes do FEC não são mais do que os símbolos de paridade do código de erros *Reed-Solomon(255,239)*. Para efeitos de correcção, a trama é dividida em várias palavras de código que são processadas independentemente. As palavras de código de uma linha da trama são entrelaçadas, sendo que os bytes de paridade são transmitidos no final de cada linha.

Os blocos de codificação e descodificação recorrem a operações aritméticas sobre $CG(256)$, que são diferentes das operações convencionais em $\mathbb{N}$ ou $\mathbb{R}$. Como as linguagens de descrição de hardware não suportam directamente operações sobre corpos de Galois, torna-se necessário implementar funções capazes de realizar tais operações. Nesta secção começa-se por apresentar a forma de implementar os operadores básicos sobre $CG(256)$, seguindo-se a descrição do codificador. O descodificador é um circuito bastante complexo, por isso começa-se por apresentar o seu diagrama de blocos genérico, passando depois à descrição detalhada de cada bloco.

3.7.1 Operadores em $CG(256)$

Para uma fácil utilização, estes operadores fundamentais devem ser combinatórios e com o menor atraso possível, pois serão utilizados intensivamente. Para o bloco de codificação serão necessários apenas somadores e multiplicadores. Já no descodificador será ainda necessário dividir.

Os somadores sobre corpos finitos $CG(2^m)$ podem ser facilmente implementados fazendo o XOR bit-a-bit dos operandos (secção 2.4.1). Assim, não é necessário qualquer circuito adicional para descrever a adição em VHDL. Portanto é apenas necessário descrever o funcionamento do multiplicador e do divisor.

Multiplicação em $CG(256)$

Numa operação de multiplicação binária o resultado final pode ser obtido somando certas versões deslocadas do multiplicando. Para saber que versões deslocadas adicionar é necessário analisar o multiplicador da direita para a esquerda. Se o bit menos significativo do multiplicador for '1', o multiplicando é somado ao resultado intermédio e de seguida é deslocado para a esquerda. Caso o bit seja zero, apenas se efectua o deslocamento do multiplicando [PH05]. Este procedimento é executado para todos os bits do multiplicador. No final, a multiplicação de dois operandos de n bits resultará num produto com $2n$ bits. A multiplicação em $CG(256)$ segue um procedimento semelhante, com a diferença que todas as adições são XORs bit-a-bit e que o produto de dois operandos de n bits terá de continuar a ter n bits. Considere-se o exemplo de multiplicar 189 ("10111101") por 3 ("00000011") em $CG(256)$. O resultado final terá de ter 8 bits, mas ignorando para já esta restrição obtém-se:

$$\begin{array}{r} 10111101 \\ \times\ 11 \\ \hline 10111101 \\ (\mathbf{1})0111101 \\ \hline (\mathbf{1})11000111 \end{array}$$

Como se pode verificar o resultado intermédio tem 9 bits. Considerando agora que α é uma raiz do polinómio primitivo $H(x) = x^8 + x^4 + x^3 + x^2 + 1$ (definido na secção 2.4.4), pode-se escrever que $\alpha^8 + \alpha^4 + \alpha^3 + \alpha^2 + 1 = 0$ ou $\alpha^8 = \alpha^4 + \alpha^3 + \alpha^2 + 1$ [6]. Isto significa que em $CG(256)$ um bit na posição 8 de um vector binário equivale ao valor de 8 bits "00011101". Para obter o resultado final basta somar (XOR) a constante "00011101" ao resultado intermédio, obtendo-se "11011010" (218 em decimal). Para resultados intermédios com mais bits em excesso a metodologia é semelhante, bastando calcular o peso de cada bit extra e somar ao resultado intermédio. A tabela 3.7 mostra os equivalentes em $CG(256)$ de bits nas posições 8 a 15 de vectores binários.

Vectores binários com 16 bits	Equivalente (binário)	Equivalente (decimal)
00000001 00000000	00011101	29
00000010 00000000	00111010	58
00000100 00000000	01110100	116
00001000 00000000	11101000	232
00010000 00000000	11001101	205
00100000 00000000	10000111	135
01000000 00000000	00010011	19
10000000 00000000	00100110	38

Tabela 3.7: Equivalentes em $CG(256)$ de vectores binários com 16 bits

[6] Em $CG(2^m)$ a subtracção é equivalente à adição.

Para implementar o circuito em hardware é então necessário efectuar dois passos, o primeiro corresponde a determinar o resultado intermédio de 16 bits, de seguida é necessário analisar os seus 8 bits mais significativos e quando for encontrado um bit a '1', o equivalente a esse bit é somado (XOR) aos 8 bits menos significativos do resultado intermédio. No final o resultado estará nos 8 bits menos significativos. Uma possível função em VHDL que implementa esta operação é:

```
TYPE ROM is array(integer range 8 to 15) of std_logic_vector(7 downto 0);
CONSTANT MUL_ROM:ROM:=x"1D",x"3A",x"74",x"E8",x"CD",x"87",x"13",x"26");
    FUNCTION CGMUL(A,B:std_logic_vector(7 downto 0)) return std_logic_vector is
        VARIABLE OUTPUT: std_logic_vector(15 downto 0):= x"0000";
    BEGIN
    --Calcula o resultado intermédio de 16 bits
        FOR N in 0 to 7 loop
            IF (B(N)='1') THEN
                OUTPUT((N+7) downto N):=OUTPUT((N+7) downto N) XOR A;
            END IF;
        END LOOP;
    --Converte o resultado de 16 bits para o equivalente em CG(256)
        FOR N in 8 to 15 loop
            IF (OUTPUT(N)='1') then
                OUTPUT(7 downto 0):=OUTPUT(7 downto 0) XOR (MUL_ROM(N));
            END IF;
        END LOOP;
    --Resultado final são os 8 bits menos significativos da variável
        RETURN OUTPUT(7 downto 0);
    END FUNCTION CGMUL;
```

Divisão em $CG(256)$

Tendo já implementado o multiplicador de elementos de $CG(256)$, a maneira mais simples de dividir dois números é calcular o inverso do divisor e de seguida multiplicar pelo dividendo. Assim o problema resume-se a determinar o inverso de um elemento do corpo. Como existe um conjunto limitado de elementos (256), a maneira mais simples e eficiente de calcular o inverso, é usar uma tabela com a correspondência entre o elemento e o seu inverso.

O inverso de um elemento x de um corpo de Galois é definido como sendo o valor que multiplicado por x produz o valor 1 (α^0). Utilizando a notação exponencial $(\alpha^i)^{-1} = \alpha^{-i} = \alpha^{(n-i)}$. A tabela 3.7.1 mostra o inverso de alguns elementos de $CG(256)$. Por definição o inverso do elemento '0' é '0'.

Elementos (binário)	Elementos (exponencial)	Inverso (exponencial)	Inverso (binário)
00000001	α^0	α^0	00000001
00000010	α^1	α^{254}	10001110
00000100	α^2	α^{253}	01000111
00001000	α^3	α^{252}	10101101
	$\vdots$	$\vdots$	
01000111	α^{253}	α^2	00000010
10001110	α^{254}	α^1	00000001

Tabela 3.8: Inverso de elementos de $CG(256)$

3.7.2 Codificador RS(255,239)

A interface do *Codificador RS(255,239)* é apresentada na figura 3.21 e é composta por uma entrada de 8 bits, que a cada ciclo de relógio recebe um símbolo da palavra a codificar. A transição '0'→'1' do sinal *Enable* indica o início da palavra de código. Durante os primeiros 239 símbolos recebidos o sinal *SELout* deve estar no nível lógico '0', indicando que a saída é igual à entrada (desfasada de um ciclo de relógio). Durante esta primeira fase, os símbolos de entrada são processados até que após 239 ciclos de relógio o cálculo dos símbolos de paridade fica completo. De seguida a entrada *SELout* é colocada a '1', para os símbolos de paridade começarem a ser debitados para o canal. Após 255 ciclos de relógio todos os símbolos foram transmitidos e a entrada *SELout* deve ser colocada a '0'.

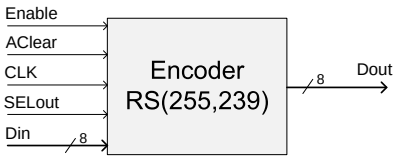


Figura 3.21: Interface do *Codificador RS(255,239)*

O circuito interno do codificador tem de ser capaz de calcular o resto da divisão de dois polinómios, de acordo com a equação 2.3 ($R(x) = D(x)\ mod\ G(x)$). O dividendo corresponde à entrada do bloco, onde cada símbolo representa um coeficiente do polinómio $D(x)$. Já o divisor é constante e é designado por polinómio gerador do código, que tem como raízes 16 potências consecutivas do elemento primitivo do corpo ($\alpha = 2$). Para efectuar a divisão polinomial é então necessário determinar os coeficientes de $G(x)$, segundo as especificações da norma, o que é conseguido desenvolvendo a equação 2.1 com o seguinte código *MATLAB*:

```
alfa = gf([1 2],8);
Gx = gf([1 1],8);
for i = 1:15
    Gx=conv(Gx,alfa.^i);
end
```

Ou em alternativa utilizando a instrução `rsgenpoly(255,239,285,0)`, onde "285" é a representação decimal do polinómio binário primitivo do corpo (equação 2.2). Ambas as formas dão origem ao polinómio:

$$G(x) = x^{16} + 59x^{15} + 13x^{14} + 104x^{13} + 189x^{12} + 68x^{11} + 209x^{10} + 30x^9 +$$

$$8x^8 + 163x^7 + 65x^6 + 41x^5 + 229x^4 + 98x^3 + 50x^2 + 36x + 59 \tag{3.1}$$

que está de acordo com [WY05].

O circuito para calcular o resto da divisão polinomial baseia-se LFSR [Ins02], como esquematizado na figura 3.22, onde as constantes $g_i (i = 0 \ldots 15)$ são os coeficientes de grau i do polinómio $G(x)$. O funcionamento do circuito é explicado em pormenor em [Roc07a].

No esquema 3.22, os sinais de selecção dos MUXs são ambos controlados pelo sinal de entrada *SELout*. Quando o codificador está a debitar os símbolos de paridade para o canal, a saída do MUX2 vai a zero, o que significa que o circuito passa a ser um simples *shift register* que está ligado à saída. Para um correcto funcionamento, a separação mínima entre palavras de entrada é $n - k = 16$ ciclos de relógio.

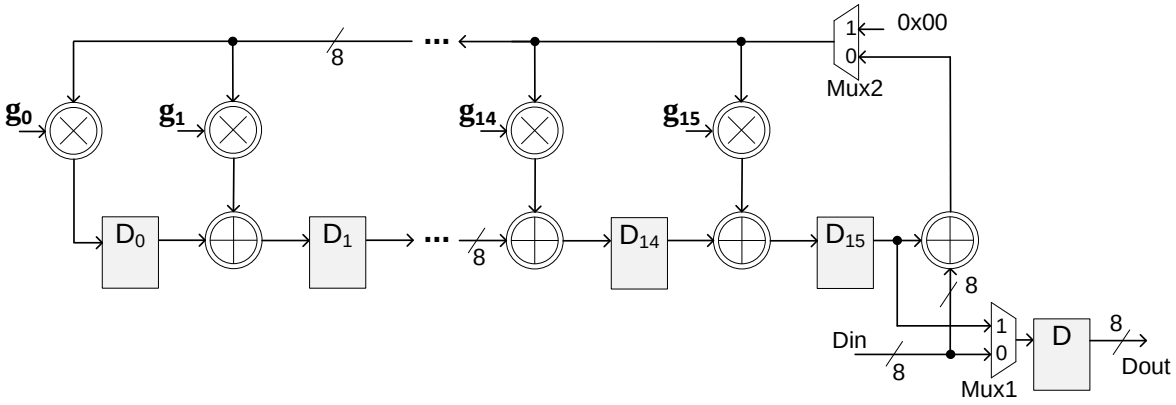


Figura 3.22: Esquema do *Codificador RS(255,239)*

3.7.3 Síntese do Codificador RS(255,239)

A tabela 3.9 mostra os resultados da síntese do bloco de codificação utilizando a ferramenta ISE. O número de *flip-flops* é menor do que seria de esperar, pois as multiplicações por $g_0 \ldots g_{15}$ dão origem a circuitos combinatórios e como tal não são necessários registos para armazenar as constantes.

Codificador RS(255,239)	
Número de *flip-flops*	148 (<1%)
Número de LUTs	243 (<1%)
Frequência máxima	$231MHz$

Tabela 3.9: Resultados da síntese do *Codificador RS(255,239)* para o FPGA LX330T

O codificador é usado apenas para simulação e como tal a sua frequência máxima de operação não é um aspecto crítico. Ainda assim a frequência máxima de relógio cumpre os requisitos temporais do sistema.

3.7.4 Diagrama de blocos genérico do descodificador $RS(255, 239)$

O método geral de descodificar códigos lineares passa por calcular o síndroma $S(x)$ da palavra recebida e associar a esse síndroma uma palavra de erro de peso mínimo $E(x)$, através de uma tabela/memória. A palavra de erro correspondente ao síndroma é então subtraída à palavra recebida, obtendo-se a informação descodificada sem erros. Este procedimento é útil para códigos de pequena dimensão, mas para códigos como o $RS(255, 239)$ a memória necessária para o descodificador seria demasiado grande. Torna-se por isso necessário explorar as propriedades dos códigos cíclicos para efectuar uma descodificação algébrica, sem recurso a memórias.

Devido à grande utilização dos códigos Reed-Solomon, existem várias técnicas de descodificação algébrica para este tipo de códigos [RL00]. Uma arquitectura comum a diversas implementações do descodificador, está esquematizada na figura 3.23. Esta arquitectura baseia-se no cálculo dos síndromas, que são usados para resolver a equação-chave (*Key Equation*). De seguida são aplicados os métodos de *Chien* e *Forney* para determinar a localização e o valor dos erros respectivamente. Para corrigir os erros, cada valor de erro determinado é adicionado (em módulo 2) à palavra recebida, que tem de ser armazenada num buffer. O bloco mais

complexo do descodificador de erros é a resolução da equação-chave, existindo diversas formas de o implementar. Os algoritmos mais utilizados são o algoritmo de *Euclides* [Cla02] e o algoritmo *Berlekamp-Massey* [WY05]. Neste projecto é utilizada uma versão melhorada do algoritmo *Berlekamp-Massey*, descrito em [SS01].

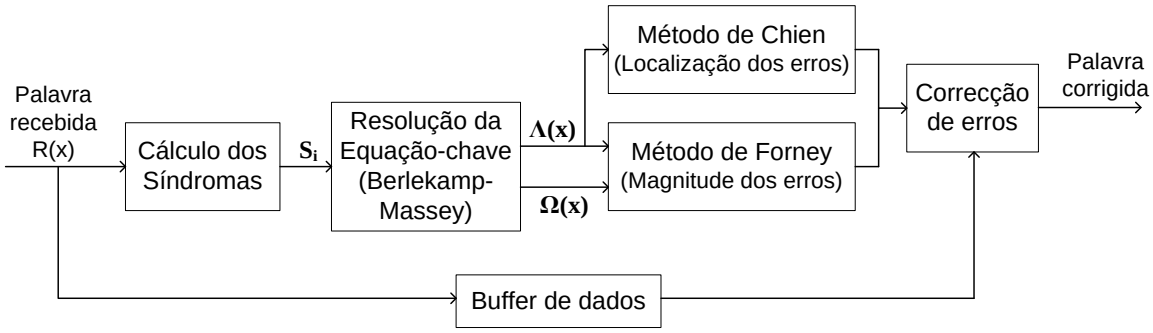


Figura 3.23: Diagrama genérico do descodificador $RS(255, 239)$

Para uma descodificação algébrica é conveniente definir os símbolos da palavra de código como polinómios. Seja $T(x)$ o polinómio da palavra transmitida e $E(x)$ a palavra de erro introduzida pelo canal, então a palavra recebida $R(x)$ é definida por

$$R(x) = T(x) + E(x) \tag{3.2}$$

onde o polinómio $E(x)$ pode ser escrito como

$$E(x) = Y_1 x^{i_1} \ + \ Y_2 x^{i_2} \ + \ \cdots \ + \ Y_v x^{i_v}.$$

A representação do polinómio $E(x)$ indica que v erros de magnitude $Y_1, Y_2, \ldots, Y_e$ ocorreram nas posições $X_1 = x^{i_1}, X_2 = x^{i_2}, \ldots X_v = x^{i_v}$. A tarefa do descodificador é determinar o polinómio de erro $E(x)$ a partir da palavra recebida $R(x)$ e depois corrigir os erros subtraindo $E(x)$ a $R(x)$. Se o número de erros v for inferior ou igual a t (definido na secção 2.4.4), o descodificador conseguirá corrigir os erros da palavra com êxito. O primeiro passo da descodificação é, como já foi dito, o cálculo dos síndromas.

3.7.5 Cálculo dos síndromas

Como visto na secção 2.4.3, a palavra de código transmitida é sempre um múltiplo do polinómio gerador $G(x)$, que por outras palavras significa que a divisão de $T(x)$ por $G(x)$ dá resto zero. Esta propriedade estende-se a cada factor do polinómio gerador. Deste modo, o primeiro passo para a descodificação é dividir a palavra recebida $R(x)$ por cada um dos $2t$ factores $(x + \alpha^i)$ de $G(x)$, originando os síndromas S_i. O resultado de cada divisão dá origem a um quociente e um resto, ou seja,

$$\begin{aligned} \frac{R(x)}{x+\alpha^i} &= Q_i(x) + \frac{S_i}{x+\alpha^i} \\ S_i &= (x+\alpha^i)Q_i(x) + R(x) \qquad (i = 0, \ldots, 2t-1) \end{aligned} \tag{3.3}$$

Substituindo $x = \alpha^i$ na equação 3.3, a expressão de cada síndroma simplifica para

$$\begin{aligned} S_i &= R(\alpha^i) \\ &= R_{n-1}(\alpha^i)^{n-1} \ + \ R_{n-2}(\alpha^i)^{n-2} \ + \ \ldots \ + \ R_1\alpha^i \ + \ R_0 \end{aligned} \tag{3.4}$$

Onde $R_{n-1}\ \ldots\ R_0$ são os símbolos da palavra recebida. Isto significa que, alternativamente, cada síndroma pode ser obtido calculando o valor do polinómio $R(x)$ para $x = \alpha^i$ [Cla02].

A expressão 3.4 simplifica o cálculo dos síndromas, pois deixa de ser necessário efectuar divisões. Um circuito simples capaz de implementar 3.4 é apresentado na figura 3.24.

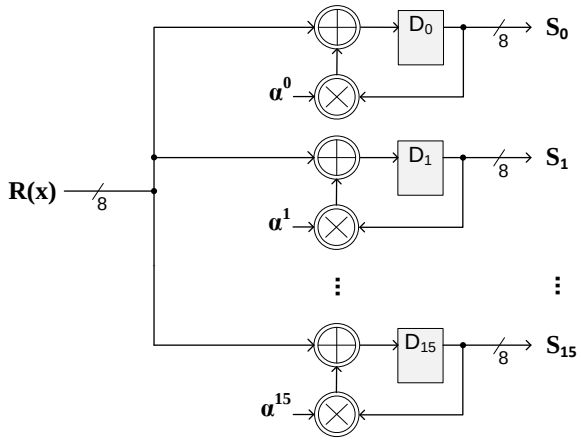


Figura 3.24: Arquitectura para o cálculo dos síndromas [WY05]

Para verificar que o circuito 3.24 efectivamente implementa as operações pretendidas, considere-se um exemplo com $R(\alpha^i) = R_3(\alpha^i)^3 + R_2(\alpha^i)^2 + R_1(\alpha^i) + R_0$. Usando a equação 3.4 para calcular o síndroma S_1, tem-se que $S_1 = R_3\alpha^3 + R_2\alpha^2 + R_1\alpha + R_0$. Analisando agora as operações efectuadas pelo circuito 3.24, pode-se escrever que S_1 é dado pela expressão 3.5 [7]. Após simplificação obtém-se 3.6 que é efectivamente o valor de S_1 pretendido.

$$\begin{aligned} S_1 &= (((R_3 + 0)\alpha + R_2)\alpha + R_1)\alpha + R_0 \qquad (3.5)\\ &= ((R_3\alpha + R_2)\alpha + R_1)\alpha + R_0 \\ &= (R_3\alpha\alpha + R_2\alpha + R_1)\alpha + R_0 \\ &= R_3\alpha\alpha\alpha + R_2\alpha\alpha + R_1\alpha + R_0 \\ &= R_3\alpha^3 + R_2\alpha^2 + R_1\alpha + R_0 \qquad (3.6) \end{aligned}$$

Para verificar algumas propriedades dos síndromas, pode-se associar as equações 3.4 e 3.2. Mas como $x = \alpha^i$ é uma raiz de $T(x)$, então $T(\alpha^i) = 0$. Portanto

$$\begin{aligned} S_i &= R(\alpha^i) = T(\alpha^i) + E(\alpha^i) \\ &= E(\alpha^i). \qquad (3.7) \end{aligned}$$

Isto significa que o valor do síndroma apenas depende da palavra de erro e não é afectado pelos dados. Além disso, quando não existem erros, todos os síndromas são zero [Cla02]. Caso isto não se verifique, os síndromas serão usados para resolver a equação-chave [8].

3.7.6 Algoritmo *Berlekamp-Massey* (BM)

Supondo que v erros, $0 \leqslant v \leqslant t$, ocorreram nas posições desconhecidas $i_1, i_2, \ldots, i_v$, então o polinómio que representa a palavra de erro é definido por

$$E(x) = e_{i_1}x^{i_1} + e_{i_2}x^{i_2} + \cdots + e_{i_v}x^{i_v}, \qquad (3.8)$$

[7] No início todos os síndromas são colocados a zero.

[8] Chamada de *Key Equation* na literatura inglesa.

onde e_{i_l} é a magnitude do erro número l. Aplicando à equação 3.8 a propriedade 3.7, o síndroma S_1 pode ser definido por $S_1 = E(\alpha) = e_{i_1}\alpha^{i_1} + e_{i_2}\alpha^{i_2} + \cdots + e_{i_v}\alpha^{i_v}$. Para simplificar a notação, pode-se fazer a mudança de variável $Y_l = e_{i_l}$ com $l = 1, 2, \ldots, v$, para representar a magnitude dos erros e $X_l = \alpha^{i_l}$ com $l = 1, 2, \ldots, v$, para representar a localização dos erros. O síndroma S_1 é então dado por

$$S_1 = Y_1X_1 + Y_2X_2 + \cdots + Y_vX_v. \tag{3.9}$$

Generalizando para os restantes síndromas obtém-se $2t$ equações com v localizações desconhecidas $(X_1, X_2, \ldots, X_v)$ e v magnitudes desconhecidas $(Y_1, Y_2, \ldots, Y_v)$:

$$S_j = Y_1X_1^j + Y_2X_2^j + \cdots + Y_vX_v^j,\ j = 1, 2, \ldots, 2t. \tag{3.10}$$

Devido à forma como os síndromas são definidos, este conjunto de equações tem pelo menos uma solução. Pode-se demonstrar que a solução é única se $0 \leqslant v \leqslant t$ [Bla83].

A resolução do sistema de equações não lineares 3.10 é complicada para valores de t elevados. Um método alternativo de resolução passa por definir um passo intermédio. Seja $\Lambda(x)$ o polinómio localizador de erros (*Error Locator Polynomial*) definido por

$$\begin{aligned} \Lambda(x) &= \prod_{l=1}^{v}(1 - X_lx) \\ &= 1 + \lambda_1x + \lambda_2x^2 + \cdots + \lambda_vx^v, \end{aligned} \tag{3.11}$$

que tem como raízes o inverso da localização dos erros (X_l^{-1}, com $l = 1, 2, \ldots, v$). O polinómio $\Lambda(x)$ pode ser usado para resolver o sistema de equações 3.10, como descrito em [WB94]. No entanto continua a ser um método pouco eficiente, pois a dimensão da matriz formada a partir de 3.11 não é conhecida à partida e tem de ser determinada por tentativa e erro [Bla83]. Um método mais eficiente passa por usar o algoritmo *Berlekamp-Massey* para calcular o polinómio $\Lambda(x)$.

O ponto de partida do algoritmo *Berlekamp-Massey* (BM) é a identidade de Newton [Moo05] que é dada por

$$S_j = -\sum_{i=1}^{v} \lambda_iS_{j-i}, \quad j = v + 1, v + 2, ..., 2t. \tag{3.12}$$

A fórmula 3.12 descreve a saída de um LFSR com coeficientes $\lambda_1, \lambda_2, \ldots, \lambda_v$. Para que a fórmula esteja correcta, é necessário determinar os coeficientes λ_i de modo a que o LSFR gere a sequência de síndromas $S_1, S_2, \ldots, S_{2t}$.

No algoritmo BM, o LFSR que gera a sequência dos síndromas é construído de modo iterativo. Partindo do LFSR capaz de produzir S_1, o próximo passo é verificar se o mesmo LFSR é capaz de gerar o conjunto $\{S_1, S_2\}$. Se conseguir não são feitas modificações, caso contrário é determinado um novo LFSR. As modificações são feitas de modo a que o novo LFSR tenha sempre a menor dimensão possível, deste modo garante-se que no final o LFSR produz a sequência $\{S_1, S_2, \ldots, S_{2t}\}$ e os seus coeficientes correspondem ao polinómio localizador de erros $\Lambda(x)$ de menor grau [Moo05].

Como o novo LFSR é determinado com base em cálculos anteriores, é necessária uma notação para representar $\Lambda(x)$ em diferentes estados do algoritmo. Seja L_k o tamanho do LFSR produzido na iteração k do algoritmo, então $\Lambda^{[k]}(x) = 1 + \lambda_1^{[k]}x + \cdots + \lambda_{L_k}^{[k]}x^{L_k}$ é o polinómio intermédio na iteração k. Supondo que numa iteração intermédia o polinómio

$\Lambda^{[k-1]}(x)$, de tamanho L_{k-1}, produz a sequência de síndromas $\{S_1, S_2, \ldots, S_{K-1}\}$, então para saber se também gera S_k é necessário calcular

$$\hat{S}_k = -\sum_{i=1}^{L_{k-1}} \lambda_i^{[k-1]} S_{k-i}. \tag{3.13}$$

Se $\hat{S}_k$ for igual ao valor S_k conhecido, então não é necessário ajustar o LFSR, caso contrário existirão discrepâncias δ_k diferentes de zero, associadas a $\Lambda^{[k-1]}(x)$:

$$\delta_k = S_k - \hat{S}_k = S_k + \sum_{i=1}^{L_{k-1}} \lambda_i^{[k-1]} S_{k-i} = \sum_{i=0}^{L_{k-1}} \lambda_i^{[k-1]} S_{k-i}.$$

Neste caso o novo polinómio localizador de erros tem de ser alterado, como descrito em [Moo05] até que após a iteração final todas as discrepâncias serão zero ($S_i - \hat{S}_i = 0$).

Após a determinação de $\Lambda(x)$ é necessário calcular o polinómio avaliador de erros (*Error Evaluator Polynomial*) $\Omega(x)$, que está relacionado com $\Lambda(x)$ e com $S(x)$ através da equação-chave [9]:

$$\Omega(x) = [S(x)\Lambda(x)] \; mod \; x^{2t}, \tag{3.14}$$

onde o polinómio síndroma $S(x)$ é definido por

$$\begin{aligned} S(x) &= \sum_{j=0}^{2t-1} S_j x^j \\ &= S_0 + S_1 x + S_2 x^2 + \cdots + S_{2t-1} x^{2t-1}. \end{aligned}$$

A operação módulo x^{2t} na equação 3.14 equivale a descartar todos os termos de grau $\geq 2t$ do produto $S(x)\Lambda(x)$.

Arquitectura iBM

Uma implementação prática do algoritmo BM é descrita em [RST91] e modificada em [SS01], passando a chamar-se *inversionless BM* (iBM). No algoritmo iBM não é necessário efectuar divisões, ao contrário do algoritmo BM original. Por esta razão, o algoritmo iBM é adoptado neste projecto. A figura 3.25 mostra o diagrama de blocos do algoritmo iBM.

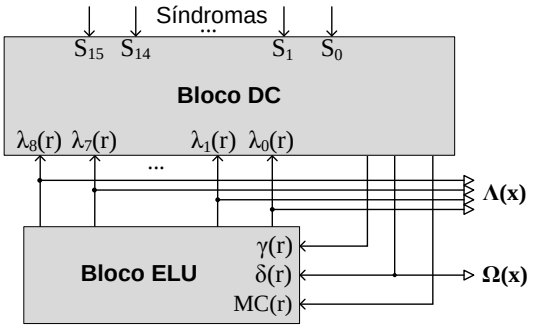


Figura 3.25: Diagrama de blocos do algoritmo iBM [SS01]

[9] Se o polinómio síndroma for definido por $S(x) = S_1 x + S_2 x^2 + \cdots + S_{2t} x^{2t}$, então a equação-chave é definida por $\Omega(x) = [(1 + S(x))\Lambda(x)] \; mod \; x^{2t+1}$.

O bloco DC (*Discrepancy Computation*) da figura 3.25 contém registos para armazenar os síndromas S_i, operadores sobre corpos finitos para calcular as discrepâncias $\delta(r)$ e a unidade de controlo de toda a arquitectura. Encontra-se ligado ao bloco ELU (*Error Locator Update*), que contém registos para armazenar $\Lambda(r)$ e $B(r)$, bem como operadores sobre corpos finitos para actualizar os polinómios. Em cada ciclo de relógio, o bloco DC calcula as discrepâncias $\delta(r)$ e passa este valor juntamente com $\gamma(r)$ para o bloco ELU, que é responsável por actualizar os polinómios.

O algoritmo iBM é descrito na figura 3.26 sob a forma de pseudo-código. Este algoritmo é subdividido em 4 passos, que são executados pelos blocos DC e ELU. O passo iBM.1 é calculado pelo bloco DC, enquanto os passos iBM.2 e iBM.3 são executados em paralelo pelo bloco ELU. Após a execução dos 3 primeiros passos, o cálculo do polinómio localizador de erros $\Lambda(x)$ fica concluído. De seguida o bloco DC é reutilizado para executar o passo iBM.4, pois as operações necessárias nos passos iBM.1 e iBM.4 são muito semelhantes. Este último passo permite calcular o polinómio avaliador de erros $\Omega(x)$.

```
Inicialização: λ0(0) = b0(0) = 1.  λi(0) = bi(0) = 0,  i = 1, 2, ..., t.  k(0) = 0.  γ(0) = 1.
Entradas:      Si,  i = 0, 1, ..., 2t − 1.
    for r = 0 step 1 until 2t − 1 do:
        begin
            Passo iBM.1: δ(r) = Srλ0(r) + Sr−1λ1(r) + ··· + Sr−tλt(r).
            Passo iBM.2: λi(r + 1) = γ(r)λi(r) − δ(r)bi−1(r),  i = 0, 1, ..., t.
            Passo iBM.3: if δ(r) ≠ 0 and k(r) ≥ 0 then
                             bi(r + 1) = λi(r),    i = 0, 1, ..., t.
                             γ(r + 1) = δ(r).
                             k(r + 1) = −k(r) − 1.
                         else
                             bi(r + 1) = bi−1(r),    i = 0, 1, ..., t.
                             γ(r + 1) = γ(r).
                             k(r + 1) = k(r) + 1.
                         end
        end
    for i = 0 step 1 until t − 1 do:
        Passo iBM.4: ωi(2t) = Siλ0(2t) + Si−1λ12t + ··· + S0λi(2t).
Saídas: λi(2t),  i = 0, 1, ..., t.  ωi(2t),  i = 0, 1, ..., t − 1.
```

Figura 3.26: Pseudo-código do algoritmo iBM [SS01]

3.7.7 Métodos de *Forney* e *Chien*

O próximo passo do descodificador é calcular as posições dos erros e a sua magnitude, usando os polinómios $\Lambda(x)$ e $\Omega(x)$ determinados anteriormente. Para implementar estas tarefas a forma mais eficiente é usar o métodos de *Chien* (ou *Chien search*) para determinar a posição dos erros e em paralelo usar a fórmula de *Forney* para calcular o valor dos erros. O último passo é simplesmente somar a palavra de erro ao símbolo correspondente da palavra recebida.

Método de *Chien* (*Chien search*)

O polinómio localizador de erros $\Lambda(x)$, definido pela equação 3.11, foi construído de modo a que as suas raízes sejam o inverso da localização dos erros X_l^{-1}. Um método eficiente para calcular as raízes de $\Lambda(x)$ é usar o método de *Chien* que não é mais do que calcular o valor do polinómio para todos os elementos de $CG(256)$. Se o resultado $\Lambda(\alpha^i) = 0$, com

$i = 0, 1, \ldots, n-1$, significa que foi encontrada uma raiz. Uma vez que o primeiro símbolo da palavra corresponde ao termo x^{n-1}, a primeira raiz a testar é $\alpha^{-(n-1)} = \alpha^1$. O seguinte é $\alpha^{-(n-2)} = \alpha^2$ até ao último símbolo que corresponde à raiz $\alpha^0 = \alpha^{255}$. A sequência de cálculos a realizar é a seguinte:

$$\begin{aligned}
&\Lambda(\alpha^1) = 1 + \lambda_1(\alpha^1)^1 + \lambda_2(\alpha^1)^2 + \cdots + \lambda_v(\alpha^1)^v \quad \rightarrow \texttt{Testar 1° símbolo}\\
&\Lambda(\alpha^2) = 1 + \lambda_1(\alpha^2)^1 + \lambda_2(\alpha^2)^2 + \cdots + \lambda_v(\alpha^2)^v \quad \rightarrow \texttt{Testar 2° símbolo}\\
&\quad\vdots\\
&\Lambda(\alpha^{254}) = 1 + \lambda_1(\alpha^{254})^1 + \lambda_2(\alpha^{254})^2 + \cdots + \lambda_v(\alpha^{254})^v \quad \rightarrow \texttt{Testar 254° símbolo}\\
&\Lambda(\alpha^{255}) = \Lambda(\alpha^0) = 1 + \lambda_1 + \lambda_2 + \cdots + \lambda_v \quad \rightarrow \texttt{Testar 255° símbolo}
\end{aligned}$$

A arquitectura usada para procurar as raízes de $\Lambda(x)$ é apresentada no esquema a) da figura 3.27. O circuito apresentado testa um elemento do corpo por cada ciclo de relógio. O polinómio $\Lambda(x)$ terá grau $v \leq t$, ou seja, no máximo 9 coeficientes. Nesta arquitectura considera-se que o número de coeficientes é sempre igual a 9, no entanto alguns coeficientes podem ser nulos, dependendo do número de erros. A saída $\Lambda_{impar}(\alpha^i)$ será necessária para o algoritmo de *Forney*, que é executado em paralelo.

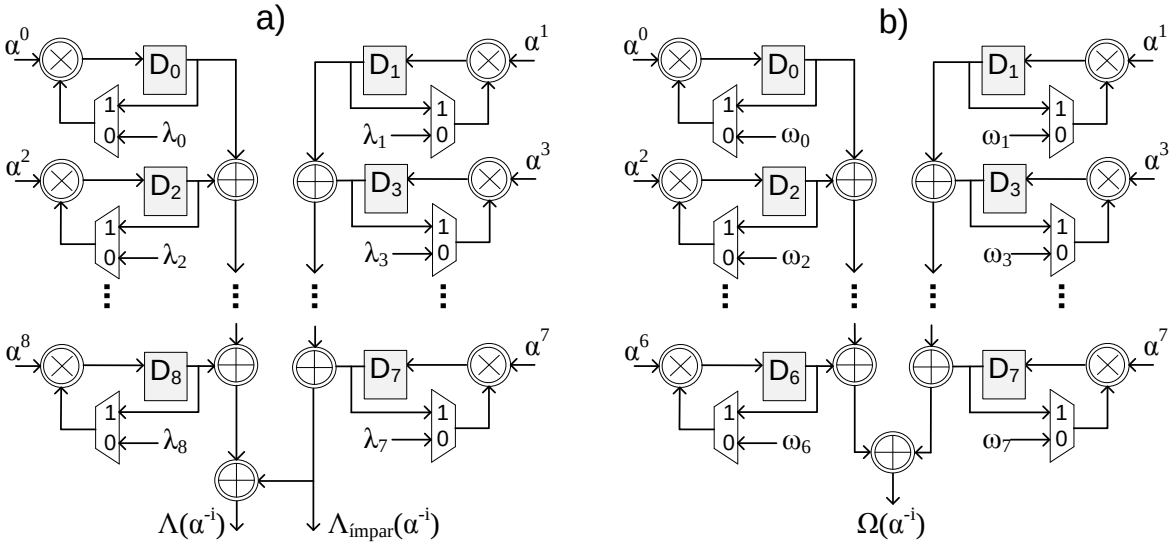


Figura 3.27: Arquitectura para implementar o método de *Chien*

No início do cálculo os MUXs seleccionam a entrada 0 para que sejam feitos os cálculos para testar o 1° símbolo. Nos símbolos seguintes os MUXs seleccionam a entrada 1 para realimentar o valor anterior e multiplicar pelas mesmas constantes, porque $\lambda_i(\alpha^2)^i = [\lambda_i(\alpha^1)^i](\alpha^1)^i$. Desta forma consegue-se verificar todos os elementos do corpo, multiplicando sempre pelas mesmas constantes, que são os primeiros 9 elementos de $CG(256)$, como apresentado na tabela 3.10. Para $\alpha^0 = 1$ o circuito de multiplicação por este valor pode ser substituído por um curto-circuito.

Algoritmo de *Forney*

Tendo determinado a localização dos erros $X_1, X_2, \ldots, X_v$ pode-se determinar o valor dos erros $Y_1, Y_2, \ldots, Y_v$ através das equações dos síndromas 3.10. No entanto existe um método muito mais eficiente, chamado de algoritmo de *Forney*. Este algoritmo usa os polinómios $\Omega(x)$ e a derivada do polinómio localizador de erros $\Lambda'(x)$. Os valores dos erros são dados por

$$Y_j = \alpha^i \frac{\Omega(\alpha^{-i})}{\Lambda'(\alpha^{-i})} \tag{3.15}$$

Constantes a multiplicar	Equivalente (binário)	Equivalente (decimal)
$(\alpha^1)^0$	00000001	1
$(\alpha^1)^1$	00000010	2
$(\alpha^1)^2$	00000100	4
$(\alpha^1)^3$	00001000	8
$(\alpha^1)^4$	00010000	16
$(\alpha^1)^5$	00100000	32
$(\alpha^1)^6$	01000000	64
$(\alpha^1)^7$	10000000	128
$(\alpha^1)^8$	00011101	29

Tabela 3.10: Constantes utilizadas no método de *Chien*

A operação de derivação de polinómios em corpos finitos segue as regras de derivação em $\mathbb{R}$. No entanto prova-se que os termos de índice par são nulos [Moo05]. Após, simplificações a derivada reduz-se a:

$$\Lambda(x) = 1 + \lambda_1 x + \lambda_2 x^2 + \cdots + \lambda_v x^v$$
$$\Lambda'(x) = \lambda_1 + \lambda_3 x^2 + \lambda_5 x^4 + \lambda_7 x^6$$

Portanto o polinómio $\Lambda'(x)$ corresponde a eliminar os termos de $\Lambda(x)$ de expoente par e dividir por x. No entanto na expressão do polinómio gerador, $b = 0$ (equação 2.1), ou seja, α^0 é uma das raízes do polinómio gerador, a soma dos termos de índice ímpar de $\Lambda(x)$ corresponde directamente a $\Lambda'(\alpha^{-i})/\alpha^i$, o que simplifica a fórmula 3.15 para

$$Y_i = \frac{\Omega(\alpha^{-i})}{\Lambda_{impar}(\alpha^{-i})}. \tag{3.16}$$

A soma dos coeficientes ímpares de $\Lambda(x)$ é obtida directamente do circuito a) da figura 3.27. O princípio de funcionamento do circuito b) é o mesmo do circuito a) e serve para calcular $\Omega(\alpha^{-i})$. Para implementar o algoritmo de *Forney* basta calcular o inverso de $\Lambda_{impar}(\alpha^{-i})$, recorrendo a uma memória, e multiplicar o resultado por $\Omega(\alpha^{-i})$. Este cálculo só é válido quando é detectada uma posição com erro, como esquematizado na figura 3.28.

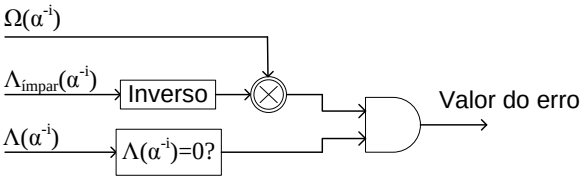


Figura 3.28: Arquitectura para implementar algoritmo de *Forney*

Os polinómios calculados pelo algoritmo iBM na realidade não são $\Lambda(x)$ e $\Omega(x)$, mas sim $\beta\Lambda(x)$ e $\beta\Omega(x)$ onde β é um valor arbitrário. No entanto este pormenor não afecta o algoritmo de *Forney* pois ao efectuar a divisão, a variável β é cancelada. Para o cálculo das raízes de $\Lambda(x)$ o valor β também não tem influência pois as raízes de $\beta\Lambda(x)$ e $\Lambda(x)$ são exactamente as mesmas.

Correcção de erros

Depois de obter os valores dos erros e a sua localização, a correcção propriamente dita é simplesmente a soma do valor do erro ao respectivo símbolo da palavra recebida. Para este efeito é necessário armazenar a palavra de código num buffer como esquematizado no diagrama geral do descodificador, figura 3.23. A dimensão do *buffer* corresponde à latência do descodificador.

3.7.8 Síntese do Descodificador FEC

A tabela 3.11 mostra os resultados da síntese dos diversos blocos que constituem o *Descodificador FEC*. Como esperado o algoritmo iBM é o bloco que utiliza mais recursos e também o bloco que permite uma menor frequência, no entanto ainda superior à frequência de operação ($167MHz$). O bloco de correcção é essencialmente um *buffer* e por isso permite uma frequência de operação muito elevada. É também apresentado o resultado da síntese do descodificador completo.

Descodificador FEC	N° de flip-flops	N° de LUTs	Frequência máxima
Cálculo dos Síndromas	139 (<1%)	227 (<1%)	$223MHz$
Algoritmo BM	487 (<1%)	1421 (<1%)	$203MHz$
Métodos de *Forney* e *Chien*	153 (<1%)	302 (<1%)	$211MHz$
Correcção de erros	338 (<1%)	96 (<1%)	$479MHz$
Descodificador completo	1109 (<1%)	2016 (≈1%)	$202MHz$

Tabela 3.11: Resultados da síntese do *Descodificador FEC* para o FPGA LX330T

O descodificador completo utiliza aproximadamente 1% das LUTs disponíveis no FPGA. Quanto aos *flip-flops*, apenas são necessários cerca de 0,5% do total. Verifica-se que a soma dos recursos de cada bloco isolado é ligeiramente superior ao total necessário para implementar o descodificador completo. Isto significa que na implementação do descodificador completo, são efectuadas algumas optimizações pela ferramenta de síntese. A frequência máxima de operação do bloco de descodificação é aproximadamente igual à frequência máxima do Algoritmo BM.

3.8 Sistema Integrado

Após a síntese de cada bloco em separado, é necessário ligar os módulos de acordo com a figura 3.5, para construir o receptor. Na tabela 3.12 é apresentado o resultado da síntese do sistema integrado. Como se pode observar, os requisitos temporais são cumpridos, com uma frequência máxima próxima dos $200MHz$.

Sistema Integrado	
Número de *flip-flops*	15 437 (≈7,4%)
Número de LUTs	32 192 (≈15,5%)
Frequência máxima	$206MHz$

Tabela 3.12: Resultados da síntese do *Sistema Integrado* para o FPGA LX330T

A área ocupada pelo sistema é em grande parte devida aos 16 blocos de descodificação do FEC utilizados pelo sistema. Cada bloco de descodificação ocupa ligeiramente menos de

1% (tabela 3.11) da área do dispositivo. Por isso mesmo, o total de LUTs necessárias para implementar o receptor ronda os 15,5% do total disponível (207 360). Em relação à área ocupada, os restantes blocos têm uma contribuição pouco significativa.

3.9 Conclusão

Neste capítulo é abordada a implementação dos blocos para efectuar o alinhamento de octetos, detecção de sincronismo, *descrambler* e descodificação do código de erros FEC. Dentro do bloco de sincronismo é efectuada a monitorização mínima do tráfego, através da análise dos principais campos do cabeçalho OTN. Após a descrição, cada bloco é sintetizado para verificar qual a frequência máxima de operação e quais os recursos necessários para implementação no FPGA LX330T da família *Virtex-5* da Xilinx. Por último os módulos são ligados para construir o receptor e é mostrado o resultado da síntese.

Devido à sua complexidade, o bloco descodificador de erros é dividido em vários blocos funcionais, que são descritos em detalhe. Os blocos que compõem o descodificador são sintetizados separadamente e depois ligados para formar o descodificador completo.

O próximo passo é simular cada bloco individualmente. Após a verificação de cada bloco, é necessário interligar todas as partes para verificar se o sistema integrado funciona como pretendido.

Capítulo 4

Simulação do sistema

Neste capítulo são validados os blocos descritos no capítulo anterior. A validação é efectuada recorrendo a simulações na ferramenta *ModelSim*. Nas simulações são usados *test-benches* que em VHDL são componentes sem portos, usados para definir os estímulos de entrada e caso seja necessário, as ligações entre blocos. A estrutura genérica do *test-bench* está ilustrada na figura 4.1. O programa *ModelSim* é usado para visualizar as formas de onda dos sinais de entrada e saída.

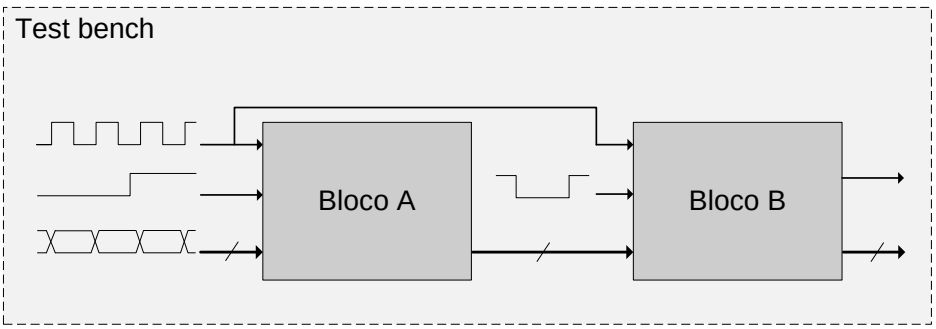


Figura 4.1: Utilização de um *test bench* para simulação

Usando as ferramentas *ModelSim* e ISE, é possível fazer simulações tendo em conta os atrasos dos circuitos (*Post-Translate Simulation*, *Post-Map Simulation*, *Post-Route Simulation*). Este tipo de simulações são demoradas e difíceis de analisar. Por esta razão, neste projecto são realizadas apenas simulações comportamentais (*Behavioral Simulation*). Estas simulações servem para verificar se o comportamento do circuito é o esperado, no entanto não permitem determinar qual a frequência máxima de operação, ao contrário das simulações com atrasos. A frequência máxima de operação é determinada pela ferramenta ISE, como indicado no capítulo anterior.

Nas secções seguintes são analisados os resultados das simulações de cada um dos blocos criados. Por último é simulado o sistema com todos os blocos integrados.

4.1 Scrambler/Descrambler

O bloco de *Scrambler* gera uma sequência de bits pseudo-aleatória, que é somada (XOR) aos dados de entrada. Para recuperar os dados originais basta somar novamente a mesma sequência, ou seja, o bloco de *Scrambler* e *Descrambler* são iguais. Uma maneira simples de testar o funcionamento do *Scrambler/Descrambler*, é ligar dois blocos iguais em série e

verificar se a saída é igual à entrada. Devido ao atraso interno de cada bloco, a saída estará atrasada dois ciclos de relógio, relativamente à entrada. Na figura 4.2 está esquematizado o circuito usado para verificar o funcionamento do bloco *Scrambler*.

O sinal de activação do *Descrambler* (*Descrambler_EN*) é atrasado um ciclo de relógio, para compensar o atraso interno do *Scrambler* e garantir que ambos os blocos iniciam o seu funcionamento no mesmo ponto da trama. Para teste foi usada uma entrada com um padrão fácil de reconhecer. A entrada toma os valores 0x1111, 0x2222, ..., 0xFFFF e repete-se indefinidamente.

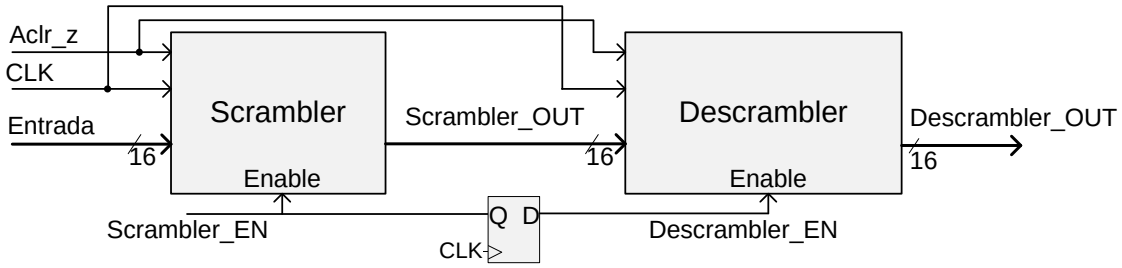


Figura 4.2: Arquitectura para simulação do *Scrambler*

No diagrama temporal da figura 4.3, o sinal *Scrambler_EN* é colocado a '0' durante 3 ciclos de relógio para simular a recepção da sequência de alinhamento. É possível verificar que o sinal de saída (*Descrambler_OUT*) é sempre igual à entrada atrasada dois ciclos de relógio. No entanto verificam-se duas situações distintas. Quando o sinal de *Scrambler_EN* está a '0', os blocos estão transparentes e a entrada passa directamente para a saída, ou seja, as saídas *Scrambler_OUT* e *Descrambler_OUT* coincidem com a entrada. Isto acontece nos instantes iniciais e no intervalo $[40 \ldots 55]ns$. No resto do tempo, os blocos estão activos e a saída do *Scrambler* (actual) passa a ser igual à operação XOR entre a entrada (anterior) e a sequência (anterior) gerada pelo LFSR. Quando o *Scrambler* está desactivado, a sequência gerada pelo LFSR (*Sequência_LFSR*) é inicializada em 0xFFFF, que será o primeiro valor a ser somado aos dados na próxima activação do *Scrambler*.

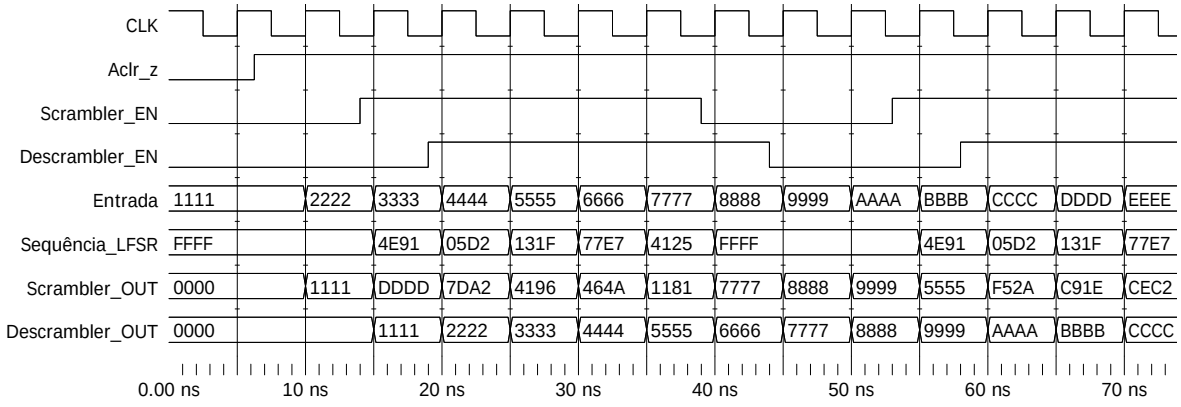


Figura 4.3: Simulação em *ModelSim* do comportamento do *Scrambler*

O resultado da simulação da figura 4.3 permite verificar o funcionamento do circuito quando este está em modo transparente e activo, no entanto não garante que a sequência gerada pelo LFSR (sinal *Sequência_LFSR*) é correcta. Uma maneira de verificar é gerar a mesma sequência em *MATLAB* e comparar os resultados. De seguida é apresentada uma forma de gerar a sequência pretendida em *MATLAB* [Kim03]:

```
S=[16 12 3 1 0];
n=S(1); n2=S(2); n3=S(3); n4=S(4);
R=ones(1,n);
L=2^n-1;
for i=1:L       %Gerar a sequência binária
    LFSR_bin(i)=R(n);
    R=[mod(R(n)+R(n2)+R(n3)+R(n4),2) R(1:n-1)];
end
j=1;
for i=1:4:L-4  %Converter a sequência binária para hexadecimal
    LFSR_hex(j)=dec2hex(bi2de(LFSR_bin(i:i+3),'left-msb'));
    j=j+1;
end
```

No código anterior, o vector S contém os expoentes do polinómio gerador da sequência $1 + x + x^3 + x^{12} + x^{16}$, que é definido na norma G.709 (OTN). O primeiro ciclo `for` do código gera a sequência binária, enquanto o segundo ciclo `for` converte a sequência para hexadecimal. Após uma breve comparação de resultados, é possível verificar que os primeiros valores da sequência obtida em *MATLAB* (vector *LFSR_hex*) são 0xFFFF 4E91 05D2 131F 77E7 4125, o que está de acordo com a simulação da figura 4.3 (sinal *Sequência_LFSR*).

4.2 Alinhador de Octetos

Tal como visto em secções anteriores, o controlo do bloco de alinhamento é feito pelo bloco *Detector de Sincronismo*. Assim para verificar o funcionamento do bloco *Alinhador de Octetos* é também necessário utilizar o bloco *Detector de Sincronismo*. Adicionalmente é usado o bloco *Scrambler* para simular a codificação dos dados no transmissor. O circuito utilizado para a simulação está esquematizado na figura 4.4. Para esta simulação, o bloco *Detector de Sincronismo* apenas é usado para controlar o *Alinhador de Octetos*, como tal apenas estão representadas algumas das suas entradas e saídas. As restantes funcionalidades são testadas mais à frente. O bloco com nome *Introdução de Offset* permite introduzir um desfasamento aos dados de entrada.

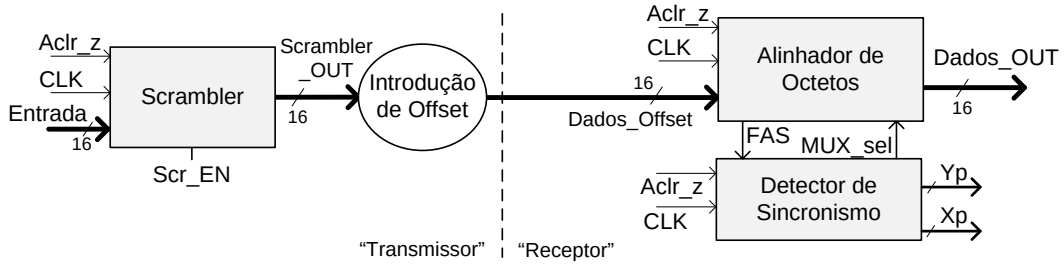


Figura 4.4: Arquitectura para simulação do *Alinhador de Octetos*

O controlo do sinal *MUX_sel* depende da máquina de estados de detecção de OOF. Se o sistema está no estado *ST_OOF*, significa que o FAS não foi detectado na posição correcta, nessa situação *MUX_sel* é incrementado de duas em duas tramas recebidas. Quando o FAS é detectado, o sistema deixa de estar em *ST_OOF* e por isso o sinal *MUX_sel* é mantido. Na figura 4.5 é apresentado o diagrama temporal onde é possível verificar o comportamento do sinal *MUX_sel*. O sinal *Entrada* pretende simular a trama OTN, mas com um tamanho reduzido para facilitar a visualização dos sinais. Neste exemplo a trama possui 4 linhas mas apenas 80 colunas. O sinal *Numero_trama* é gerado pelo *test bench* apenas para facilitar a

análise do diagrama. Tal como na trama OTN original, os primeiros 6 bytes do sinal *Entrada* são a sequência FAS. Pode-se observar que enquanto o FAS não é detectado, a máquina de estados está em *ST_OOF* e portanto a sinal *MUX_sel* é incrementado de duas em duas tramas. Quando o sinal *MUX_sel*=3 o FAS é detectado pela primeira vez e o estado passa para *ST_TO_IF*. A partir deste momento o sinal *MUX_sel* é mantido e como se pode verificar o FAS passa a ser detectado nas tramas seguintes.

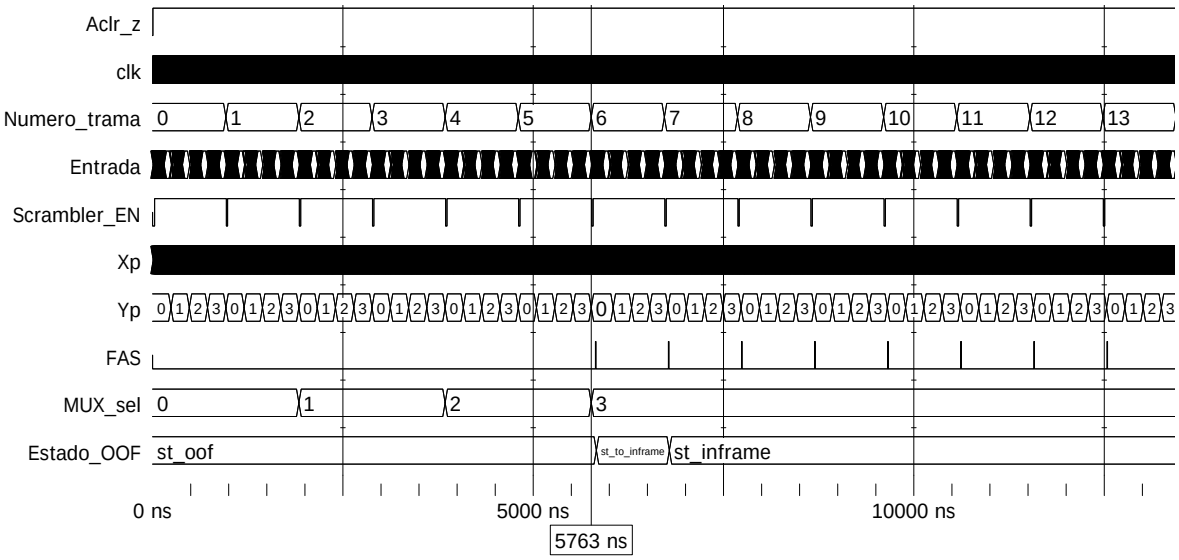


Figura 4.5: Simulação do comportamento geral do bloco *Alinhador de Octetos*

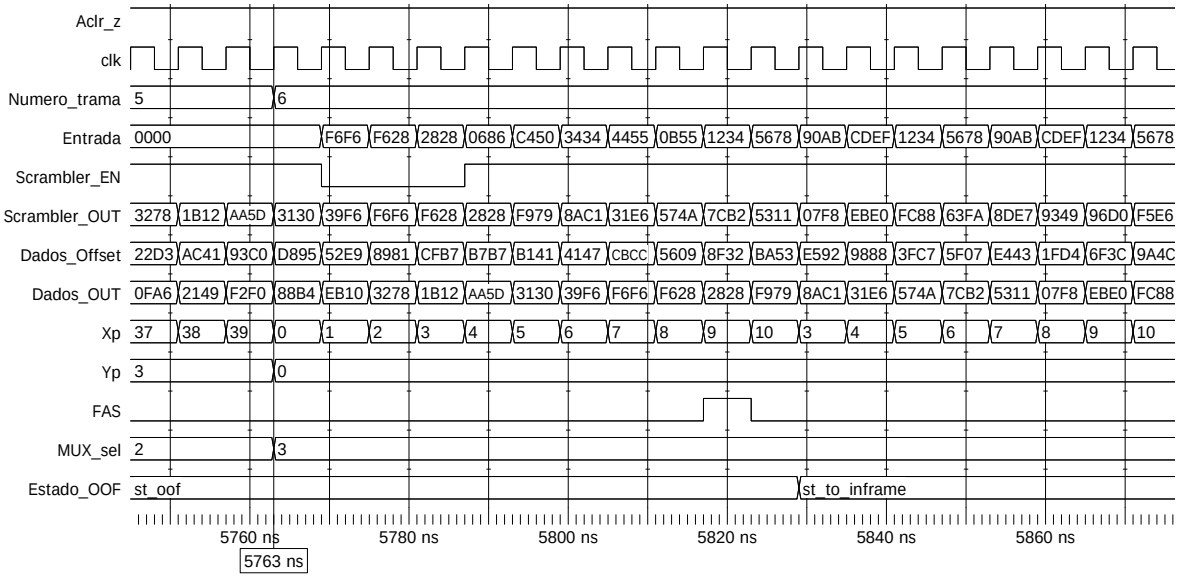


Figura 4.6: Simulação do *Alinhador de Octetos* quando o FAS é detectado

O diagrama temporal da figura 4.6 é a ampliação do diagrama da figura 4.5, no momento em que o FAS é detectado pela primeira vez. É possível observar que a sequência FAS aparece de forma evidente no sinal *Scrambler_OUT* e após a introdução do offset (sinal *Dados_offset*), deixa de ser possível identificar a sequência facilmente. No sinal de saída do bloco alinhador, *Dados_OUT*, a sequência FAS aparece novamente de forma evidente, ou seja, com o valor

actual de *MUX_sel* o alinhamento está correcto. De notar também que quando o estado é *ST_OOF*, as coordenadas Xp e Yp são incrementadas, mas podem não ser os valores correctos, já que neste estado ainda não foi identificado o início de trama. Quando a máquina de estados transita de *ST_OOF* para *ST_TO_IF*, a coordenada Xp é actualizada de Xp=10 para Xp=3. A coordenada Yp mantém o valor Yp=0, pois já estava correcta.

4.3 Detector de Sincronismo

Na figura 4.7 está esquematizado o circuito usado para simular o bloco de detecção de sincronismo. Esta arquitectura é usada para simular várias situações, onde o que varia é o tipo e localização dos erros introduzidos. Mais uma vez é usado o *Scrambler* para simular a codificação dos dados no transmissor. Esta arquitectura permite também verificar o controlo do *Descrambler*, presente na parte de recepção. Neste exemplo não é introduzido nenhum offset, por isso o bloco *Alinhador de Octetos* apenas serve para sinalizar a detecção do FAS.

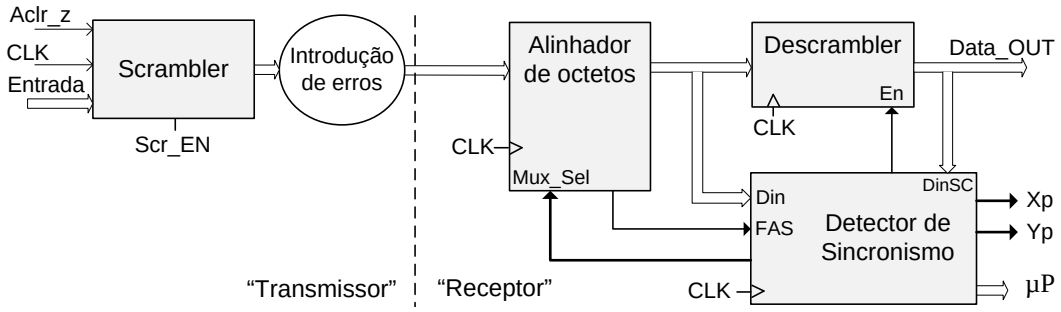


Figura 4.7: Arquitectura para simulação do *Detector de Sincronismo*

Detecção de OOF

Esta simulação pretende verificar o funcionamento da máquina de estados de detecção de OOF. Na figura 4.8 é apresentado o diagrama temporal onde se pode ver o comportamento da máquina de estados, na presença de erros nos bits da sequência de alinhamento da trama. Nas primeiras tramas não há erros e o FAS é detectado correctamente, por isso a máquina transita do estado inicial (*ST_OOF*) para *ST_TO_IF* e após a segunda trama correcta, transita para *ST_IF*. Quando o sistema passa para *ST_IF*, o sinal OOF (*Out Of frame*) passa para '0', indicando que o alinhamento está correcto.

A certa altura são introduzidos erros na sequência de alinhamento das tramas número 3 e 4. Nesta situação o FAS deixa de ser detectado durante duas tramas e a máquina passa para *ST_TO_OOF*, onde existe um contador de tramas *TramasNaoOk* que permite saber o número de tramas erradas consecutivas. Como após duas tramas (por volta dos $5000ns$) o FAS é detectado correctamente, a máquina volta para o estado *ST_IF*. No entanto, nas tramas seguintes o FAS volta a estar errado e após 5 tramas com FAS errado, a máquina transita para *ST_OOF*, aos $9669ns$. O sinal OOF é declarado (OOF='1') assim que a máquina entra no estado *ST_OOF*.

Ainda na figura 4.8 é possível observar que a saída do *Detector de Sincronismo* (*Detector_OUT*) tem sempre o valor 0xFFFF, o que indica que os dados podem não ser válidos. Os dados só são considerados válidos quando o FAS é detectado correctamente durante $3ms$, ou seja, quando a máquina de estados de detecção de LOF passa para o estado *ST_NORMAL*. O comportamento da máquina de estados de detecção de LOF é analisado de seguida.

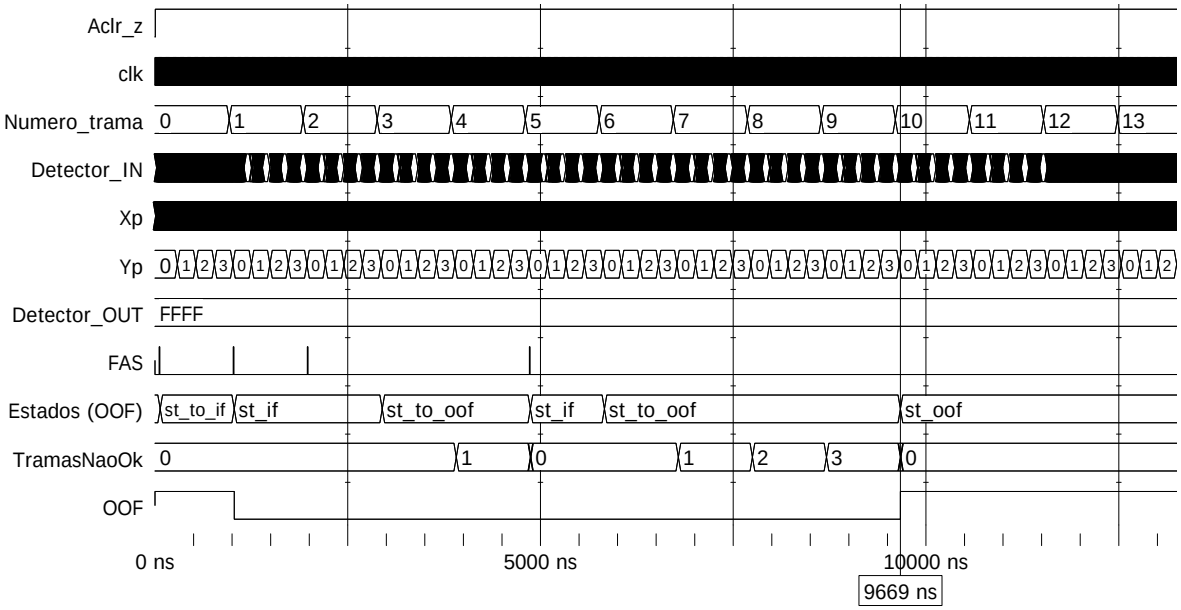


Figura 4.8: Simulação do máquina de detecção de OOF

Detecção de LOF

Na figura 4.9 é apresentado o funcionamento da máquina de estados de detecção de LOF (*Loss Of Frame*), no momento da transição para o estado *ST_LOF*. O sinal LOF é declarado sempre que o sistema entra no estado *ST_LOF*.

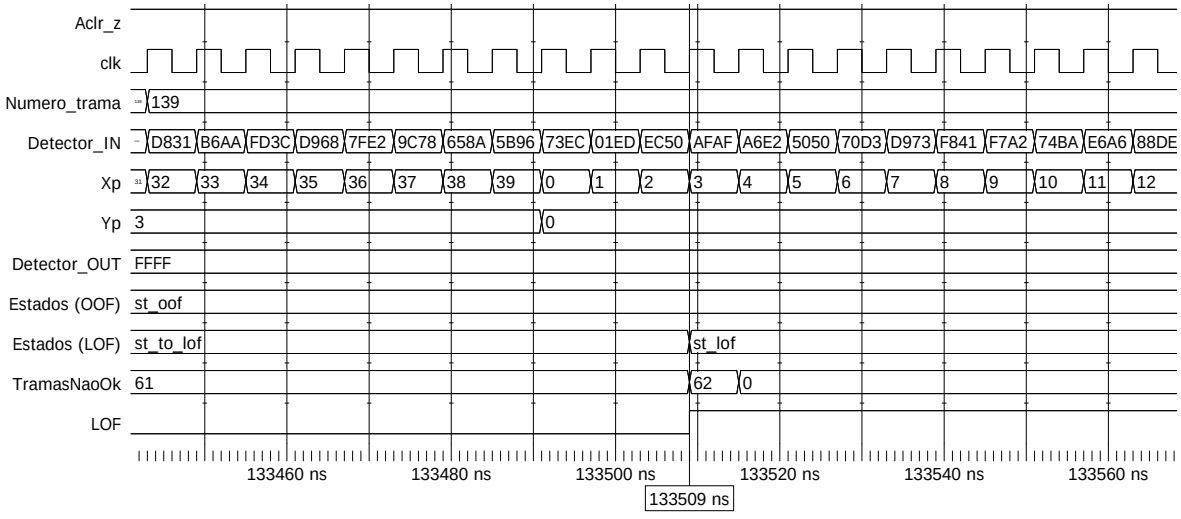


Figura 4.9: Simulação do máquina de detecção de LOF

Para verificar a detecção de LOF, inicialmente são colocadas à entrada cerca de 70 tramas com a sequência de alinhamento correcta. Como a trama tem o FAS correcto, o sistema passa rapidamente para *ST_IF*. A máquina de detecção de LOF (*Loss Of Frame*) conta o tempo que o sistema passa em *ST_IF* e após 62 tramas [1] ocorre a transição de *ST_LOF* para *ST_NORMAL*, onde o sinal LOF é limpo (LOF=‘0’). Após o sistema transitar para *ST_NORMAL*, são introduzidos erros na sequência de alinhamento em todas as tramas

[1] 62 tramas completas correspondem a $3ms$.

seguintes. Isto obriga a máquina de estados de detecção de OOF a transitar para *ST_OOF* depois de 5 tramas erradas. A máquina de estados de detecção de LOF ao verificar que o sistema está em *ST_OOF* vai transitar para *ST_TO_LOF*, onde começa a contar o número de tramas erradas. Só após 62 tramas erradas o sistema transita para *ST_LOF*, o que acontece aos 133509ns.

Detecção de OOM

Na figura 4.10 é possível observar o comportamento da máquina de estados de detecção de OOM (*Out Of Multiframe*). Neste exemplo o MFAS recebido apenas fica correcto a partir da trama número 71. Na trama seguinte o sistema transita para *ST_TO_IM* e passada outra trama, passa para *ST_IM*. Para verificar se o valor MFAS da trama está correcto, é utilizado o registo *Ultimo_MFAS* que guarda o valor MFAS da última trama recebida. Se o valor do MFAS actual da trama for igual a *Ultimo_MFAS*+1, o valor recebido está correcto.

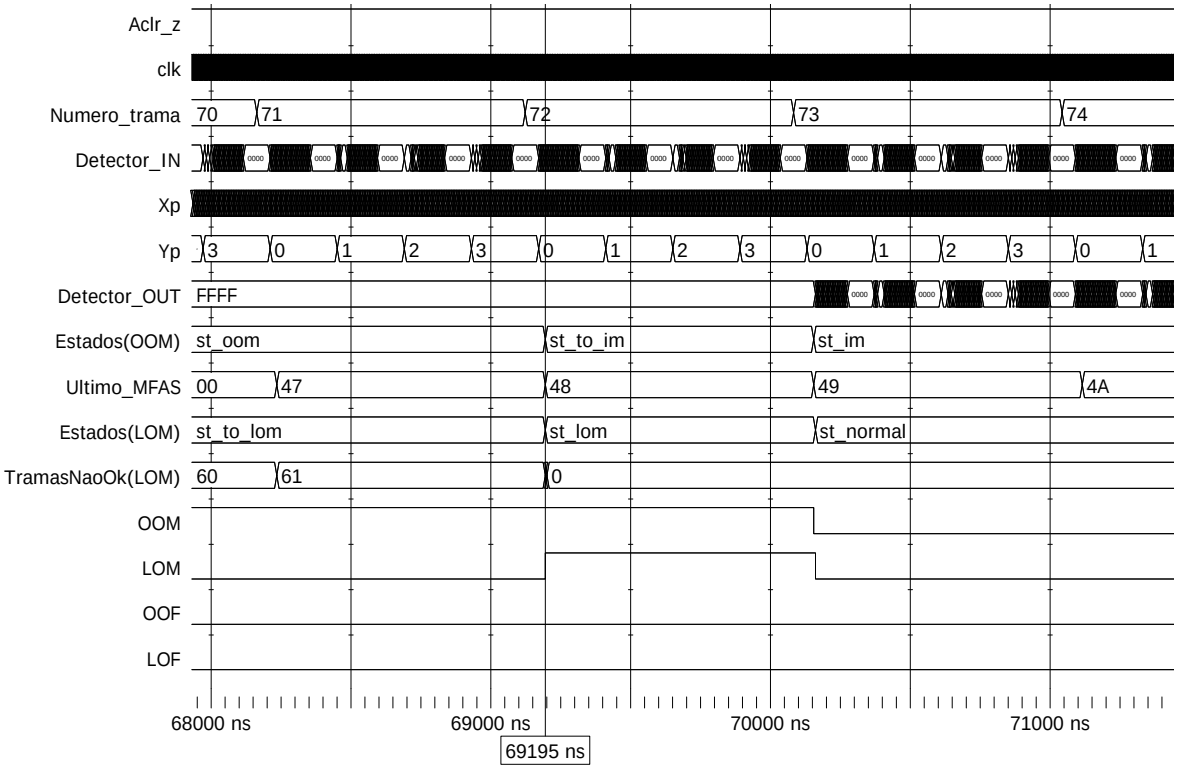


Figura 4.10: Simulação do máquina de detecção de OOM e LOM

Detecção de LOM

Para verificar o funcionamento da máquina de estados de detecção de LOM (*Loss Of Multiframe*) foram usadas tramas com o campo MFAS errado (MFAS=0x00). Isto obriga o sistema a permanecer em *ST_OOM*. Após 62 tramas erradas, a máquina de estados de detecção de LOM transita para *ST_LOM* e o sinal LOM é declarado (LOM='1'). Este comportamento é ilustrado do diagrama temporal da figura 4.10. O contador de tramas erradas (*TramasNaoOk*) permite saber durante quantas tramas consecutivas o sistema esteve em *ST_OOM*. No exemplo da figura 4.10 o sistema fica no estado *ST_LOM* apenas durante uma trama, pois a partir da trama 72 o MFAS fica correcto. Na trama número 73 o sistema

transita para *ST_IM* e como consequência, a máquina de estados de detecção de LOM passa imediatamente para o estado *ST_NORMAL*. Com a passagem para *ST_NORMAL*, o sinal LOM é limpo (LOM='0').

Ainda na figura 4.10 é possível observar que a partir da trama número 73, a saída do bloco (*Detector_OUT*) deixa de ter o valor 0xFFFF, ou seja, fica válida. Isto acontece porque todos os sinais de alinhamento (OOF, LOF, OOM e LOM) ficam a '0', indicando que os alinhamentos de trama e multi-trama estão correctos. No bloco *Detector de Sincronismo* a saída *Detector_OUT*, quando válida, é igual à entrada *Detector_IN* pois o *Detector de Sincronismo* analisa os dados da trama mas não os altera.

Campos SM e PM

O cálculo dos erros de BIP-8 dos campos SM e PM é muito semelhante, mudando apenas a localização do byte na trama. Como tal será apenas apresentada uma simulação. No caso optou-se por mostrar o exemplo de recepção e cálculo dos erros de BIP-8 do campo SM.

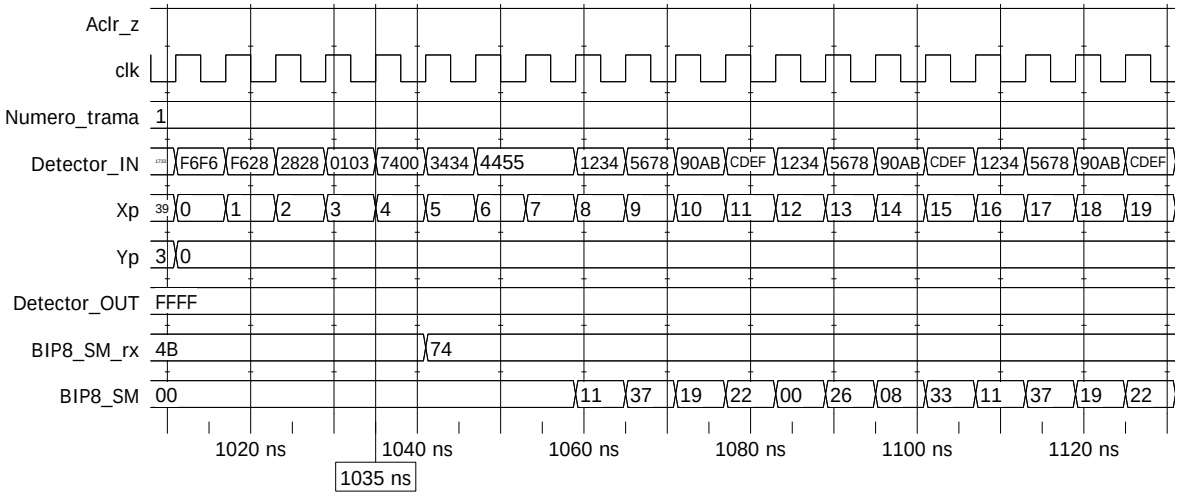


Figura 4.11: Simulação da recepção do byte BIP-8 do SM

Na figura 4.11 é apresentado o momento em que o byte BIP-8 é lido da trama e armazenado no registo *BIP8_SM_rx*. Neste exemplo, o valor de BIP-8 recebido é igual a 0x74 e corresponde aos 8 bits mais significativos da entrada (*Detector_IN*), logo após o instante 1035ns. Pode-se observar que na transição de relógio seguinte, o valor é copiado para *BIP8_SM_rx*. O cálculo de BIP-8 começa a partir de Xp=8. O registo *BIP8_SM* é usado para armazenar os resultados intermédios do cálculo de BIP-8. Pode-se verificar que o primeiro valor de *BIP8_SM* é igual à operação XOR entre o byte mais significativo e o byte menos significativo da entrada (0x11 = 0x44 XOR 0x55). O segundo valor de *BIP8_SM* é obtido: 0x37 = 0x11 XOR 0x12 XOR 0x34. Estas operações continuam até se atingir o último byte da carga-paga.

Na simulação da figura 4.12 é possível observar a parte final do cálculo de erros de BIP-8 do cabeçalho SM. O valor final é armazenado em *BIP8_SM_Latch0*. Pode-se confirmar que, após o instante 17235ns, o valor final de BIP-8 (*BIP8_SM_Latch0*) corresponde ao XOR entre os bytes *BIP8_SM*, *Detector_IN[15 ... 8]* e *Detector_IN[7 ... 0]*, anteriores a 17235ns. Ou seja, 0xA5 = 0x87 XOR 0xCD XOR 0xEF. O registo *BIP8_SM_Latch1* armazena o cálculo de BIP-8 da trama anterior e *BIP8_SM_Latch2* armazena o cálculo de BIP-8 duas tramas antes. Para verificar se houve erros, é preciso comparar o registo *BIP8_SM_Latch2* com o valor recebido (*BIP8_SM_rx*). Para comparar valores é feita a operação XOR entre *BIP8_SM_rx* e *BIP8_SM_Latch2* e o resultado é armazenado em *BIP8_SM_Calc* (0x91 =

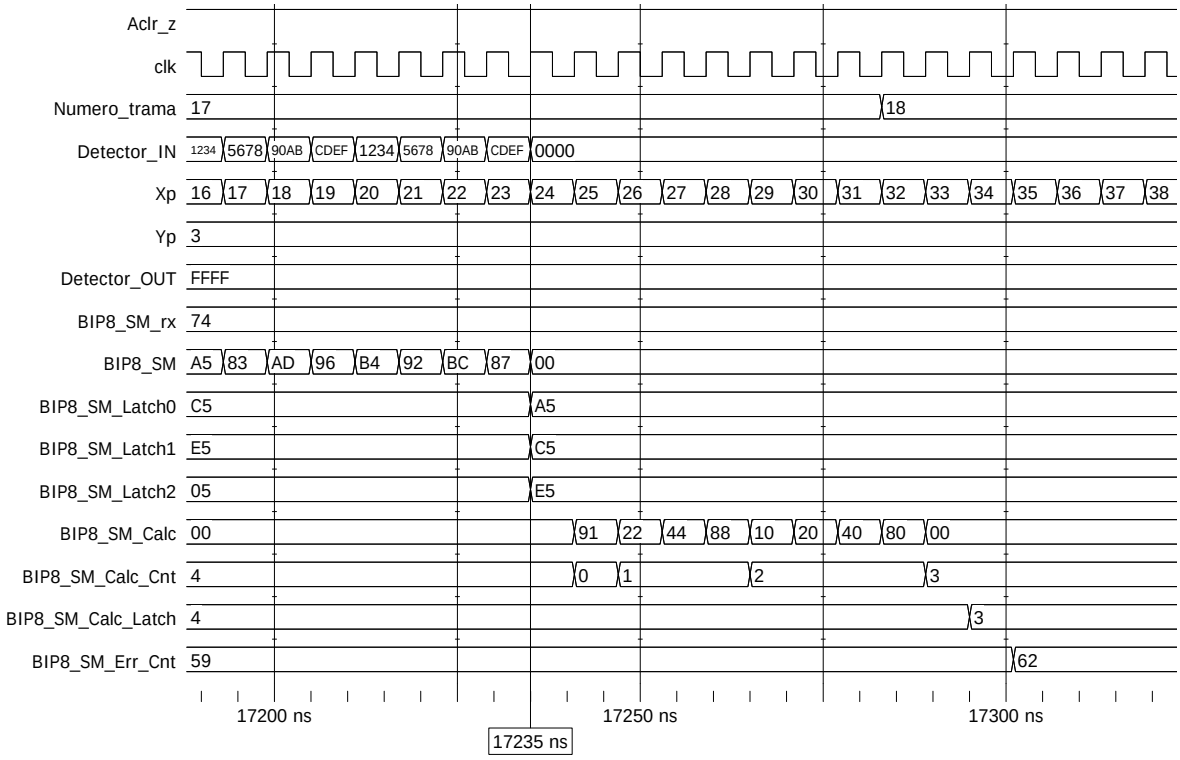


Figura 4.12: Simulação do cálculo dos erros de BIP-8 do SM

0x74 XOR 0xE5). Se o resultado da comparação for 0x00, significa que não foram detectados erros. Caso contrário, o número de erros detectados corresponde ao número de '1's do registo *BIP8_SM_Calc*.

A contagem do número de '1's de *BIP8_SM_Calc* é feita analisado o bit mais significativo do registo: se este for igual a '1' o contador *BIP8_SM_Calc_Cnt* é incrementado. No próximo ciclo de relógio o registo é deslocado para a esquerda e repete-se o processo 8 vezes, para analisar os 8 bits. No exemplo da figura 4.12 é possível verificar que no início o registo *BIP8_SM_Calc*=0x91 tem 3 bits a '1', que é efectivamente o valor final de *BIP8_SM_Calc_Cnt*. O registo *BIP8_SM_Calc_Latch* armazena o valor final de *BIP8_SM_Calc*. O contador *BIP8_SM_Err_Cnt* permite saber o número de erros de BIP-8 detectados em todas as tramas anteriores. Este contador pode ser lido ou reiniciado pelo micro processador.

Campos BEI/BIAE

A figura 4.13 ilustra o funcionamento do contador de erros de BEI e a verificação do sinal BIAE. Pode-se observar que nas primeiras tramas os 4 bits correspondentes ao campo BEI/BIAE têm um valor inferior a 9. Como tal correspondem à indicação dos erros de BIP-8 detectados pelo terminal que enviou a trama, ou seja, indicação de BEI (*Backward Error Indication*). O sinal *BEI_SM_cnt* é igual ao valor BEI recebido (*BEI_BIAE_SM_rx*) sempre que este é inferior a 9, caso contrário é igual a 0000. O registo *BEI_SM_Err_cnt* armazena o número de erros acumulado das tramas anteriores, ou seja, *BEI_SM_Err_cnt* = *BEI_SM_Err_cnt* + *BEI_SM_cnt*, como se verifica pelo diagrama temporal.

Na trama número 8, o campo BEI/BIAE passa a ter o valor 1011, o que indica que o transmissor está a enviar o sinal BIAE (*Backward Incoming Alignment Error*). Para que

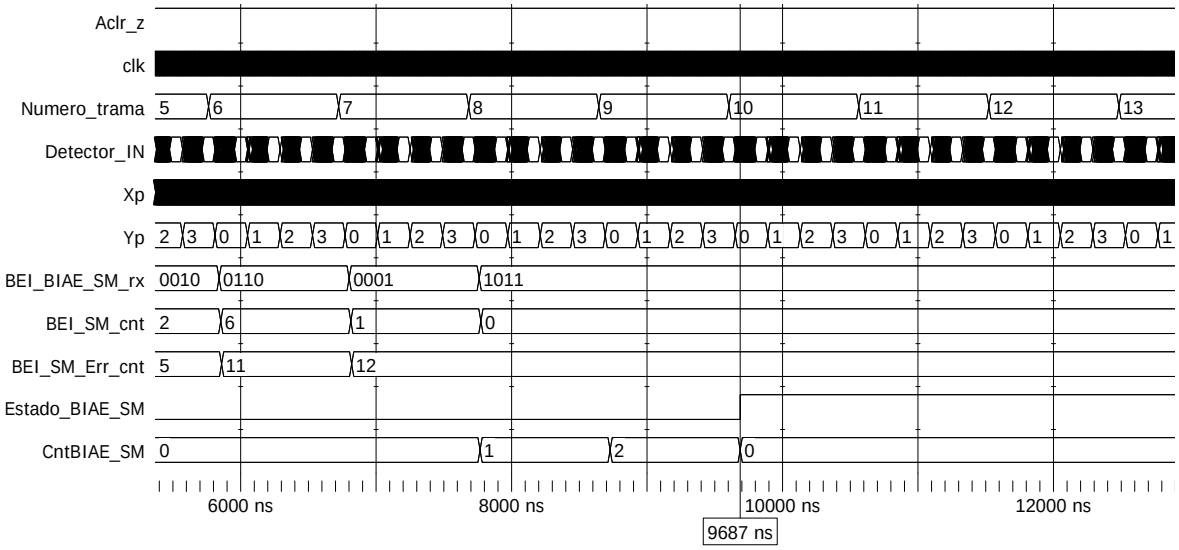


Figura 4.13: Funcionamento dos campos BEI/BIAE do SM

este sinal seja validado, tem que ser recebido durante 3 tramas consecutivas. O contador *CntBIAE_SM* permite saber quantas tramas consecutivas foram recebidas com o sinal BIAE. Após a confirmação do sinal BIAE, o sinal interno *Estado_BIAE_SM* é declarado (*Estado_BIAE_SM*=1), o que acontece aos 9687ns. Quando se recebe o sinal BIAE, o contador de erros de BEI (*BEI_SM_Err_cnt*) é mantido, pois nesta situação os bits recebidos não correspondem ao número de erros de BIP-8.

4.4 Código de erros FEC

Para verificar o descodificador do código FEC, é necessário utilizar um codificador para gerar os dados de entrada. Por isso o primeiro passo é testar o bloco de codificação para garantir que os dados de entrada estão correctos. De seguida a simulação do descodificador é feita em várias etapas. Em primeiro lugar cada bloco é testado individualmente, começando pelo bloco de cálculo dos síndromas. Os restantes blocos são introduzidos gradualmente e testados até que o *Descodificador FEC* esteja completo. O último passo é simular o descodificador integrado com o resto do sistema, sendo para isso necessário utilizar 16 blocos de descodificação a funcionar em paralelo.

4.4.1 Codificador RS(255,239)

A verificação do *Codificador RS(255,239)* é feita através da comparação dos resultados da simulação em *ModelSim* com os valores obtidos em *MATLAB*. A implementação do codificador em *MATLAB* é bastante simples, já que basta usar a função "`rsenc()`". Os parâmetros da função são a palavra a codificar, a dimensão do código e o polinómio primitivo do corpo (equação 2.2). As instruções seguintes permitem codificar os dados do vector "`msg`":

```
poly = rsgenpoly(255,239,285,0);
msg  = gf(1:239,8);
code = rsenc(msg,255,239,poly)
```

Neste exemplo a palavra a codificar (*msg*) tem 239 símbolos que tomam os valores [1, 2,. . . ,239]. Na palavra codificada (*code*) de 255 símbolos, as últimas 16 posições correspondem aos símbolos de paridade. Os 16 símbolos de paridade obtidos neste exemplo são apresentados

na tabela 4.1. O valor R_{15} corresponde ao primeiro símbolo da palavra codificada e R_0 corresponde ao último.

R_{15}	R_{14}	R_{13}	R_{12}	R_{11}	R_{10}	R_9	R_8	R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0
1	126	147	48	155	224	3	157	29	226	40	114	61	30	244	75

Tabela 4.1: Símbolos de paridade obtidos em *MATLAB*

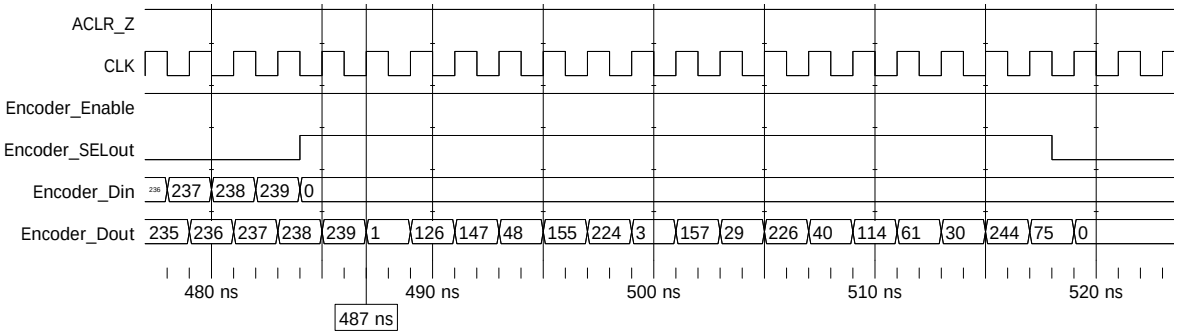


Figura 4.14: Simulação do bloco de codificação do $RS(255, 239)$

A mesma palavra de entrada foi usada para simular o bloco codificador implementado em hardware. A simulação é mostrada na figura 4.14. Os símbolos de paridade surgem após o instante $487ns$, logo depois do último símbolo de dados, que tem valor 239. Verifica-se que os símbolos de paridade obtidos são iguais aos da tabela 4.1, o que indica que o bloco estará a funcionar correctamente.

4.4.2 Descodificador RS(255,239)

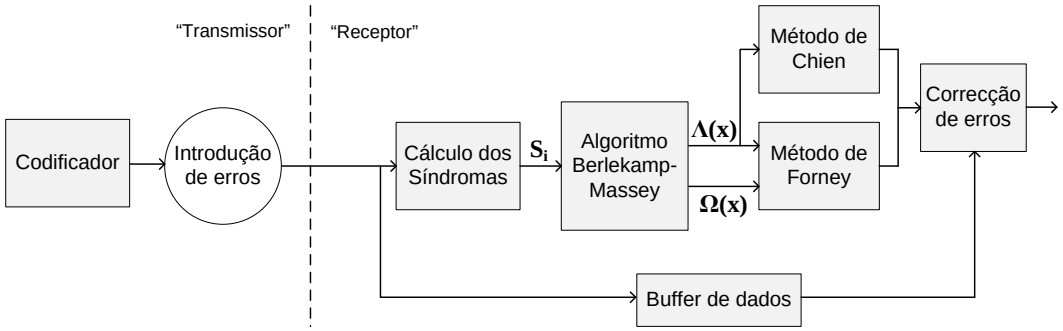


Figura 4.15: Circuito para simulação do *Descodificador RS(255,239)*

Na figura 4.15 é apresentado o circuito usado para simular o descodificador. O bloco codificador é usado para gerar os dados, que antes de entrarem no descodificador são corrompidos com erros. Para verificar os resultados obtidos em *ModelSim* são usadas as funcionalidades do *MATLAB*. Começa-se por simular o bloco de cálculo dos síndromas, de seguida são analisados os polinómios determinados pelo algoritmo *Berlekamp-Massey* e por fim são verificados os algoritmos de *Forney* e *Chien*.

Síndromas

Para simular o bloco de cálculo de síndromas, foram introduzidos erros em 8 símbolos da palavra gerada anteriormente pelo codificador. Os símbolos errados são a negação dos símbolos originais e estão nos símbolos 1, 3, 7, 99, 123, 149, 226 e 254. Uma maneira possível de calcular os síndromas em *MATLAB* é:

```
alfas = gf([1 2 4 8 16 32 64 128 29 58 116 232 205 135 19 38],8);
Si    = gf([0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0],8);
for t = 1:255
   Si = msg_erros(t) + Si.*alfas;
end
```

O vector `msg_erros` corresponde ao vector original corrompido com erros. O vector `alfas` contém as constantes que são multiplicadas pelos símbolos de dados. Estas constantes são as primeiras 16 potências do elemento fundamental do corpo $(\alpha^0, \alpha^1, \ldots, \alpha^{15})$. Os 16 síndromas obtidos são mostrados na tabela 4.2.

S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}	S_{13}	S_{14}	S_{15}
0	68	48	95	33	9	81	62	34	135	76	45	88	21	40	18

Tabela 4.2: Síndromas obtidos em *MATLAB*, para exemplo com 8 erros

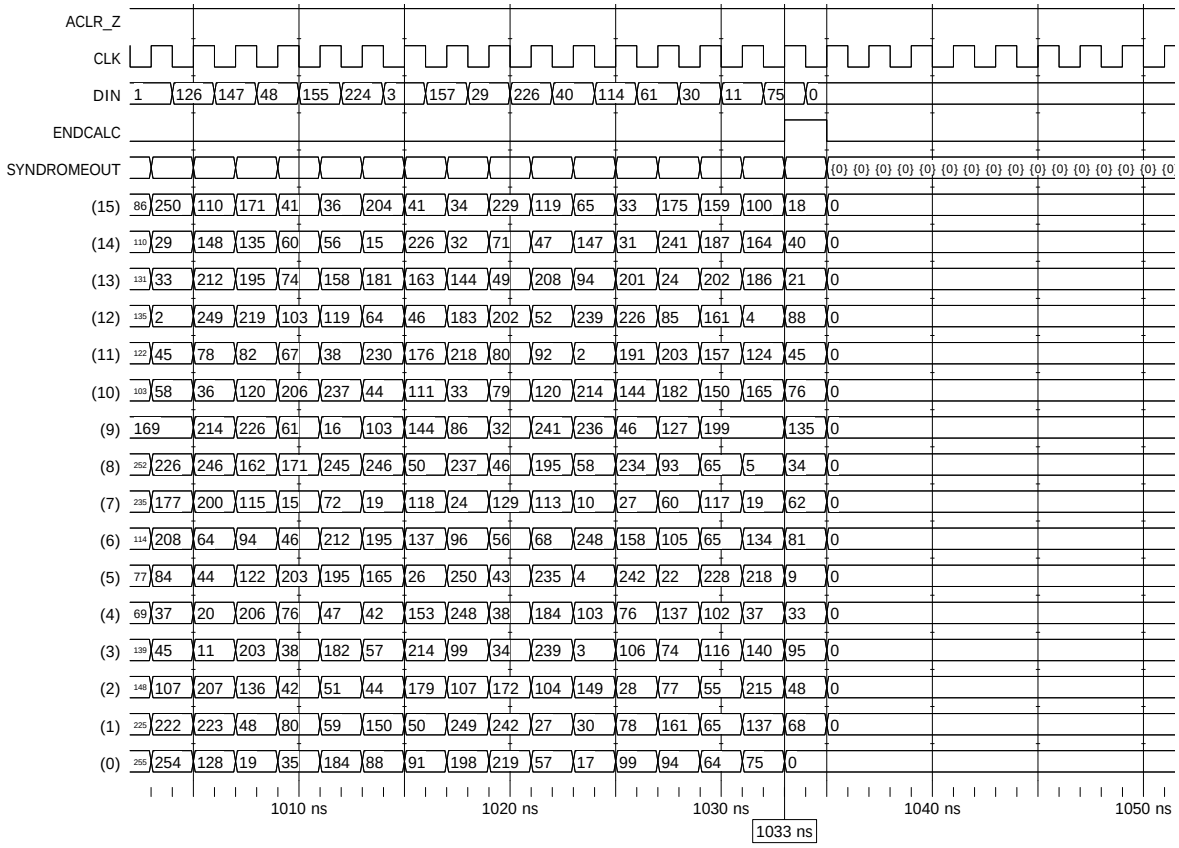


Figura 4.16: Simulação do bloco de cálculo dos síndromas

Usando a mesma palavra de entrada, a simulação obtida em *ModelSim* é apresentada na figura 4.16, onde é possível observar a evolução do cálculo dos síndromas. O final do cálculo é

indicado quando o sinal *ENDCALC* toma o valor '1'. Como se pode ver na figura 4.16 o fim do cálculo dos síndromas acontece aos $1033ns$. Verifica-se que os resultados coincidem com os valores do *MATLAB* (tabela 4.2).

Algoritmos *Berlekamp-Massey*, *Forney* e *Chien*

O algoritmo de *Berlekamp-Massey* é responsável por determinar os coeficientes do polinómio localizador de erros $\Lambda(x)$ e polinómio avaliador de erros $\Omega(x)$. Na figura 4.17 o sinal *LAMBDAOUT* possui os 9 coeficientes de $\Lambda(x)$ e *OMEGAOUT* possui os 8 coeficientes de $\Omega(x)$. O circuito de implementação do algoritmo *Berlekamp-Massey* é dividido em duas etapas. Em primeiro lugar é efectuado o cálculo do polinómio $\Lambda(x)$, que termina aos $1103ns$. De seguida o circuito é reutilizado para calcular $\Omega(x)$. O fim do cálculo de $\Omega(x)$ ocorre aos $1119ns$ e é sinalizado pelo sinal *ENDCALC*. Neste exemplo os polinómios obtidos são: $\Lambda(x) = 144x^8 + 85x^7 + 219x^6 + 31x^5 + 62x^4 + 253x^3 + 249x^2 + 216x + 47$ e $\Omega(x) = 94x^7 + 0x^6 + 39x^5 + 0x^4 + x^3 + 0x^2 + 179x + 0$.

Uma maneira de verificar se $\Lambda(x)$ está correcto é determinar as suas raízes através do seguinte código *MATLAB*:

```
lambda = gf([144 85 219 31 62 253 249 216 47],8);
locais = roots(lambda)
```

As raízes de $\Lambda(x)$ obtidas são 2, 8, 128, 72, 134, 142, 164, 197, que estão relacionadas com as posições dos erros. Analisando o sinal *ERRORFOUND* do exemplo da figura 4.17 verifica-se que ocorreram erros nas posições 1, 3 e 7. Estas posições dos erros estão relacionadas com as raízes $2 = \alpha^1$, $8 = \alpha^3$ e $128 = \alpha^7$. Ainda na figura 4.17 o sinal *ERRORDIFF* indica a magnitude do erro, que em conjunto com o sinal *ERRORFOUND* permite a correcção dos erros. Os dados corrigidos apenas aparecem na saída *DECODERDATAOUT* no ciclo de relógio seguinte.

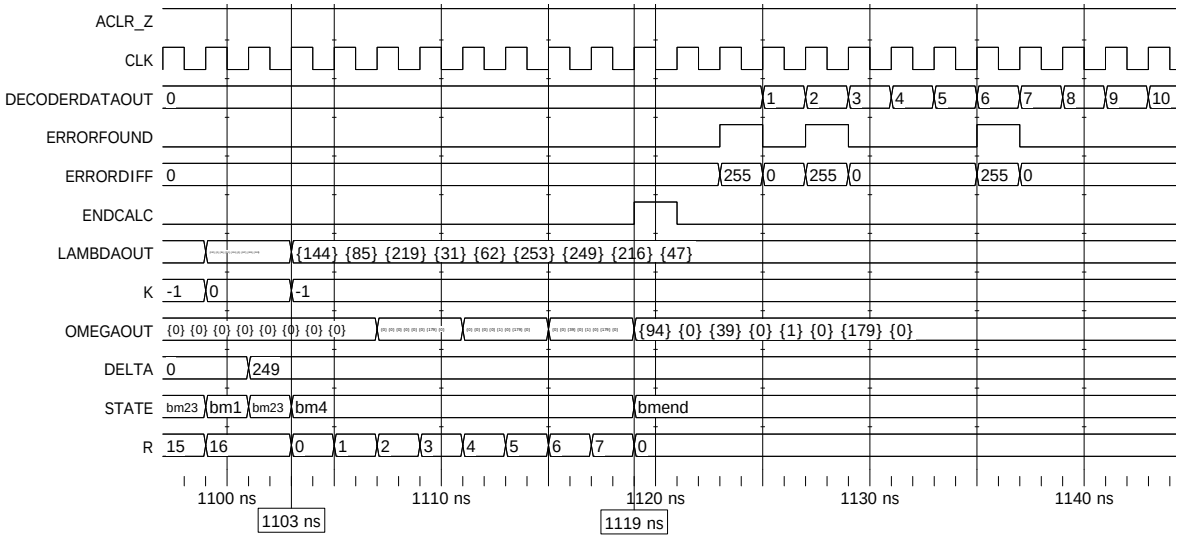


Figura 4.17: Simulação do algoritmo de *Berlekamp-Massey* e métodos de *Forney* e *Chien*

A magnitude dos erros, que na simulação da figura 4.17 tem o nome *ERRORDIFF*, é calculada recorrendo à equação 3.15, que em *MATLAB* pode ser resolvida através do seguinte código:

```
omega      = gf([94 0 39 0  1  0 179 0],8);
D_lambda   = gf([85 0 31 0 253 0 216 0],8);
```

```
alfa       = 2;
valor_erro = polyval(omega,alfa)/polyval(D_lambda,alfa)
```

Neste exemplo o valor do erro é calculado para o primeiro símbolo da palavra ($\alpha = 2$). A magnitude do erro dos restantes símbolos é obtida substituindo a variável *alfa* pelas restantes raízes de $\Lambda(x)$.

4.5 Sistema Integrado

O sistema de recepção requer todos os blocos analisados até aqui. Cada linha da trama contém 16 palavras de código entrelaçadas, por isso é necessário utilizar 16 blocos de descodificação independentes. Os blocos de descodificação operam em *pipeline*, ou seja, antes do fim do processamento de uma linha, já a linha seguinte começa a ser processada. Deste modo são necessários apenas 16 descodificadores para processar toda a trama. Na figura 4.18 é apresentado o esquema simplificado do receptor para tramas OTU1. Os dados passam por um desmultiplexer 1:16, que divide os bytes pelos 16 blocos de descodificação de erros. No final da correcção, a linha da trama sem erros é reconstruída, utilizando um multiplexer 16:1. O sistema é testado em 3 situações. No primeiro caso é introduzido um número de erros inferior à capacidade máxima de correcção, em posições consecutivas da trama. No segundo caso o número de erros excede a capacidade de correcção e por último é testado o caso particular onde é possível corrigir a maior sequência de erros consecutivos numa trama.

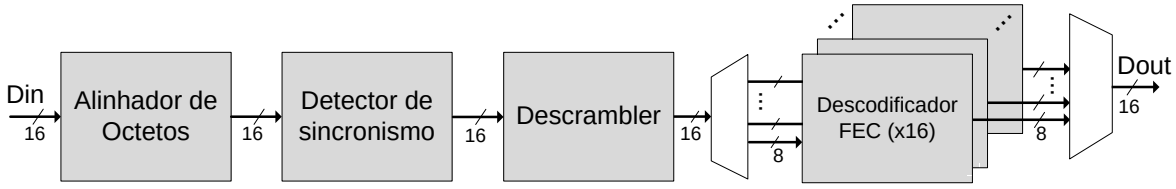


Figura 4.18: Diagrama de simulação do sistema integrado

Número de erros inferior à capacidade de correcção

Nesta simulação são introduzidos erros nos primeiros 112 bytes da carga-paga da segunda linha da trama (Yp=1), ou seja, 7 símbolos errados em cada uma das 16 palavras de código independentes. A carga-paga, antes da introdução de erros, tem todos os seus bytes com o padrão 0x11. Na figura 4.19 é possível ver o intervalo de tempo que vai desde a entrada dos dados nos descodificadores, até à saída já corrigida. Este intervalo é delimitado pelos cursores e totaliza $12010ns$. Como o sinal de relógio usado neste exemplo tem um período de $5ns$, o sistema precisa de 2402 ciclos de relógio para a descodificação completa de uma linha. De notar que um descodificador a funcionar isoladamente tem um atraso de 299 ciclos de relógio, no entanto devido ao entrelaçamento de bytes, cada descodificador apenas recebe um símbolo a cada 8 ciclos de relógio. Por esta razão o atraso total é aproximadamente 8 vezes superior. O atraso restante é devido ao multiplexer e a *flip-flops* de saída.

O tempo de latência é, em grande parte, devido ao cálculo dos síndromas que contribui com $255 \times 8 = 2040$ ciclos de relógio, correspondentes à duração duma linha completa [2] , mais um ciclo de atraso causado pelo *flip-flop* de saída. O algoritmo *Berlekamp-Massey* demora $42 \times 8 = 336$ ciclos de relógio e os métodos de *Chien* e *Forney* provocam um atraso

[2]Cada linha tem 4080 bytes, mas o barramento tem 16bits, por isso são processados dois bytes por cada ciclo de relógio.

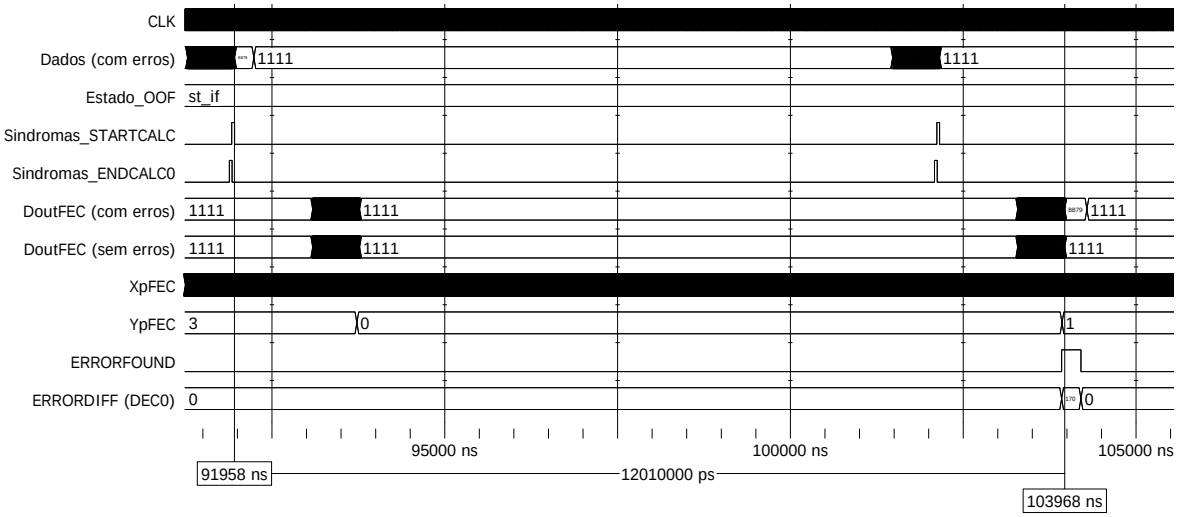


Figura 4.19: Simulação do sistema com número de erros inferior ao máximo

de $2 \times 8 = 16$ ciclos de relógio. Por último o multiplexer de saída demora $1 \times 8 = 8$ ciclos de relógio, mais um ciclo de atraso devido ao *flip-flop* de saída.

Na figura 4.19 é possível verificar que a correcção é feita com sucesso. No sinal corrigido, *DoutFEC (sem erros)*, após os $103968ns$ os primeiros bytes da carga-paga passam a ter o padrão inicial (0x11), o que não acontece com os dados antes da correcção (*DoutFEC (com erros)*).

Número de erros superior à capacidade de correcção

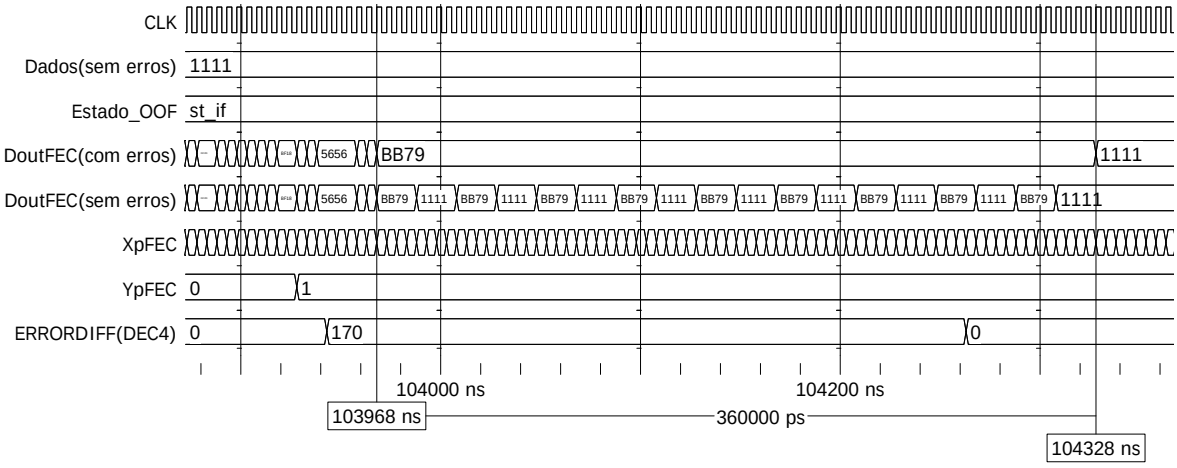


Figura 4.20: Simulação do sistema com número de erros superior ao máximo

Neste exemplo pretende-se verificar que o descodificador não consegue corrigir mais do que 8 erros por palavra de código. Para isso é efectuada uma simulação semelhante à anterior, mas com um total de 136 erros consecutivos numa linha. Isto significa que os 8 primeiros descodificadores recebem 9 erros cada e os restantes 8 descodificadores recebem 8 erros. A simulação deste exemplo é mostrada na figura 4.20, onde a duração dos erros é delimitada pelos cursores. O instante $103968ns$ é o momento onde se inicia a correcção dos erros, tal como acontece no exemplo anterior. Verifica-se que na saída corrigida (*DoutFEC(sem erros)*)

só os bytes pertencentes a palavras de código com 8 erros são corrigidas e ficam com valor 0x11. Os bytes restantes não são corrigidos e permanecem com o valor 0xBB e 0x79. Neste exemplo é possível ver que a correcção apenas tem sucesso em bytes alternados da linha. Isto acontece devido ao entrelaçamento de bytes.

Número máximo de erros consecutivos numa trama

O código FEC da norma OTN permite corrigir no máximo 128 bytes errados consecutivos numa linha da trama, no entanto haverá casos onde os erros estão localizados no final de uma linha e no início da linha seguinte. Nesta situação, o FEC permite corrigir sequências de erros maiores. A simulação da figura 4.21 mostra a situação limite onde o código FEC permite corrigir um total de 256 bytes errados consecutivos numa trama. A extensão dos erros é delimitada pelos cursores e tem uma duração total de $640ns$, correspondente aos 256 bytes. Neste exemplo são colocados 128 erros no fim da segunda linha (Yp=1) e mais 128 erros no início da terceira linha (Yp=2).

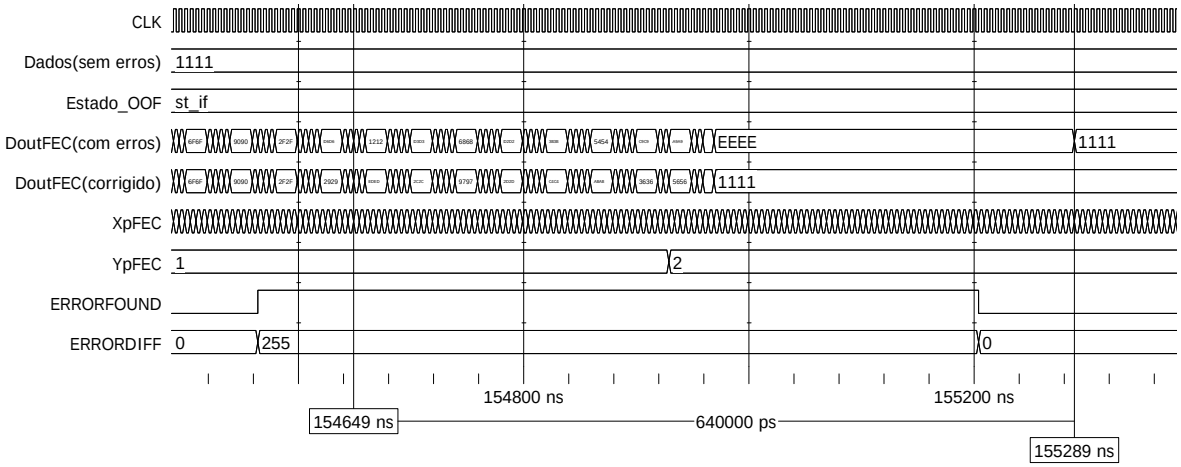


Figura 4.21: Simulação do sistema com máximo de erros consecutivos numa trama

4.6 Conclusão

Neste capítulo foi verificado o funcionamento de cada um dos blocos do sistema, através de simulações na ferramenta *ModelSim*. A verificação dos blocos mais simples fez-se através da visualização das formas de onda. Já nos blocos mais complexos foi necessário comparar com os resultados obtidos em *MATLAB*. Foi possível validar todos os blocos individualmente. No fim todos os blocos foram interligados para verificar o funcionamento do sistema completo, que também foi simulado com sucesso.

Capítulo 5

Conclusões

Com a realização deste projecto foram adquiridos conhecimentos em diversas áreas, nomeadamente em: sistemas digitais reconfiguráveis, comunicações ópticas e teoria da codificação. A implementação do receptor para redes OTN permitiu ganhar alguma experiência na utilização da linguagem de descrição de *hardware* VHDL e na simulação de hardware utilizando *ModelSim*. Apesar de o sistema não ter sido implementado em hardware, foi necessário utilizar a ferramenta de síntese ISE, para verificar se o sistema cumpre as especificações temporais e determinar quais os recursos necessários para a implementação em FPGA.

Foi feito um estudo detalhado sobre a norma G.709 (OTN), indispensável para a implementação prática dos diversos blocos do sistema. Para implementar o bloco de correcção de erros, foi também necessário estudar as bases matemáticas dos códigos de erros, começando na teoria dos corpos finitos até chegar à análise de códigos complexos como é o caso dos códigos *Reed-Solomon*.

Todos os blocos foram testados individualmente com sucesso, tal como o sistema integrado. Os resultados obtidos em *ModelSim* foram comparados com os valores esperados e nos casos mais complexos, foram comparados com simulações em *MATLAB*.

5.1 Resumo e discussão do trabalho realizado

De seguida é feito um resumo de todo o trabalho realizado, descrevendo os blocos que constituem o sistema:

- **Alinhador de Octetos** - Este bloco ocupa uma área reduzida, é constituído sobretudo por registos, um multiplexer e um comparador. Recebe um barramento de dados de 16 bits e a sua função é detectar a sequência de alinhamento (FAS) e garantir que o primeiro byte do FAS aparece no byte mais significativo do barramento de dados. Este bloco funciona em conjunto com o *Detector de Sincronismo*.

- **Scrambler/Descrambler** - É usado no transmissor para dar características aleatórias ao sinal a enviar. Deste modo impede-se o aparecimento de longas sequências de '0' ou '1's, que dificultam a extracção do sinal de relógio. No receptor o mesmo bloco é usado para recuperar os dados originais. A implementação baseia-se no LFSR descrito na norma, mas adaptado para a situação prática onde o barramento de dados tem 16 bits. Isto obriga a usar uma cadeia de XORs bastante complexa, ainda assim os recursos usados são reduzidos e a frequência máxima de operação bastante elevada.

- **Detector de Sincronismo** - Este é um bloco muito importante já que controla todos os outros. Controla o multiplexer interno do bloco *Alinhador de Octetos*, é responsável

por controlar a activação e desactivação do *Scrambler* e define quando o *Descodificador FEC* entra em funcionamento. O *Detector de Sincronismo* também é responsável por validar o alinhamento de trama e multi-trama de acordo com a norma G.798 [IT06]. Para isso faz uso de várias máquinas de estados, que dão origem aos sinais OOF, LOF, OOM e LOM usados para validação do sincronismo. São também geradas as coordenadas da trama, onde Xp indica o número da coluna e Yp o número da linha. Através destas coordenadas facilmente se consegue saber o significado dos bits recebidos. Os cabeçalhos mais importantes são também analisados neste bloco, nomeadamente os campos SM e PM. Através da análise destes campos é possível saber se ocorreram erros na transmissão e verificar se foi recebido algum alarme. A recepção e validação dos alarmes recorre mais uma vez a máquinas de estados de acordo com a norma G.798 [IT06]. Este bloco comunica com o exterior através de um microprocessador, para passar informações sobre alarmes e número de erros detectados.

- **Codificador FEC** - Apesar de não fazer parte do sistema de recepção, este bloco foi implementado para gerar as palavras de código para testar o descodificador. O circuito codificador processa os 239 símbolos da palavra a codificar e no final debita os símbolos de paridade para o barramento de dados, de modo a formar a palavra codificada de 255 símbolos. Tanto o codificador como o descodificador fazem uso de operações sobre corpos finitos, por isso foi necessário implementar alguns operadores especiais para fazer multiplicações, adições e divisões. As operações são combinatórias, para que possam ser feitas num único ciclo de relógio.

- **Descodificador FEC** - Este bloco é dividido em vários blocos com funções específicas. Para a implementação de cada sub-bloco foram analisadas várias possibilidades e no final tentou-se escolher a melhor solução para cada bloco. A arquitectura escolhida baseia-se no cálculo dos síndromas associados à palavra recebida. Os síndromas são usados para determinar os coeficientes dos polinómios $\Lambda(x)$ e $\Omega(x)$, que permitem determinar a localização e magnitude dos erros respectivamente. Para determinar os coeficientes dos polinómios é usado o algoritmo *Berlekamp-Massey*. Este algoritmo opera de modo iterativo e é a parte mais complexa do descodificador. Os restantes blocos são já bastante optimizados e utilizados na maioria das implementações pesquisadas. As localizações dos erros e a sua magnitude são determinadas pelos métodos de *Forney* e *Chien*. Por último a correcção é feita somando (XOR) o valor do erro ao respectivo símbolo.

Uma das dificuldades de implementação do receptor para redes OTN foi garantir que os blocos cumprem os requisitos temporais. A elevada frequência de operação obriga que circuitos combinatórios complexos sejam divididos em várias etapas, separadas por registos. Esta técnica foi usada por exemplo no bloco de Alinhamento.

Devido à dimensão da trama torna-se difícil efectuar simulações. Para diminuir este problema, na maioria das simulações relacionadas com os cabeçalhos, foram usadas tramas onde a carga-paga tem dimensões mais reduzidas. Deste modo simplifica-se a visualização dos sinais de interesse. Já para a simulação do funcionamento do descodificador FEC integrado no sistema, é indispensável usar a trama com as dimensões correctas.

Um dos pontos que pode ser melhorado neste trabalho é o algoritmo *Berlekamp-Massey* usado no *Descodificador FEC*. Em alternativa podem ser usados outros métodos de modo a tentar diminuir os recursos necessários e também o tempo de processamento.

Existindo já alguns trabalhos sobre OTN, a principal contribuição deste projecto é a integração dos vários blocos que compõem o receptor para redes OTN, de modo a obter um sistema funcional.

A principal dificuldade na integração dos diversos blocos está no bloco de descodificação do FEC. Isto porque para descodificar uma linha da trama são necessários 16 blocos de descodificação independentes. Além disso, os símbolos de uma palavra estão entrelaçados, por isso é necessário garantir que todos os símbolos da mesma palavra vão para o mesmo descodificador. No final é necessário voltar a colocar os símbolos já corrigidos na posição correcta da linha. Os 16 descodificadores funcionam em *pipeline*, o que significa que no total são necessários 16 descodificadores para corrigir toda a trama. O *pipeline* permite uma redução significativa da área ocupada pelo sistema.

No sistema implementado, o bloco *Detector de Sincronismo* aparece antes do bloco de correcção de erros. Outra possibilidade seria dividir o *Detector de Sincronismo* em duas partes, a parte de sincronismo, que funciona em conjunto com o bloco *Alinhador de Octetos* e *Descrambler*, e a parte de análise dos cabeçalhos. Deste modo seria possível analisar os cabeçalhos após correcção de erros. No entanto a configuração escolhida tem alguma imunidade a erros, visto que a norma prevê que para validação de alguns campos do cabeçalho, seja necessário confirmar os valores durante algumas tramas consecutivas. Assim um erro apenas será validado se aparecer na mesma posição durante várias tramas consecutivas, o que é pouco provável.

Após a síntese do sistema integrado verifica-se que os recursos utilizados são cerca de 15,5% do total disponível, o que significa que pode ser usado um FPGA mais "pequeno" para implementar o receptor. No entanto é preciso ter em conta que um dispositivo terminal para redes OTN, além do sistema de recepção, será também composto pelo sistema de transmissão. Além disso, para obter um dispositivo de rede completamente funcional é também necessário implementar o bloco de acesso à carga-paga, que não faz parte deste projecto. A escolha do FPGA LX330T permite que os restantes blocos possam ser acrescentados mais tarde.

A frequência máxima suportada pelo receptor é de $206MHz$, o que dá alguma margem de segurança, pois a frequência de operação do sistema é de $167MHz$.

5.2 Trabalho futuro

Sendo o trabalho realizado apenas uma parte do sistema terminal para redes OTN, existe ainda muito trabalho nesta área. De seguida são apresentados alguns tópicos que poderão ser abordados no futuro:

- **Teste do sistema em FPGA** - Após a validação do sistema através de simulações, o próximo passo é sintetizar o circuito e verificar o seu funcionamento com sinais reais, utilizando um FPGA. A implementação em FPGA requer alguns cuidados adicionais, nomeadamente com os sinais de relógio e com a disposição dos vários blocos dentro do *chip*. Além disso é necessário ter em atenção a interface com o exterior.

- **Integração com o terminal de transmissão** - Um terminal para redes OTN completamente funcional é composto por um bloco de recepção e outro de transmissão. Após a implementação do terminal de transmissão, pode-se formar um terminal de comunicação bidireccional. O próximo passo será usar dois terminais bidireccionais para efectuar comunicações de dados sobre fibra óptica e verificar o seu comportamento na presença de alarmes ou erros. Com um sistema deste género será possível determinar experimentalmente a potência mínima a transmitir, com e sem FEC. Pode-se ainda determinar experimentalmente qual o ganho de codificação real introduzido pela utilização do FEC em várias situações.

- **Optimização do descodificador FEC** - O bloco de descodificação do FEC é objecto de estudo de muitos investigadores, que tentam encontrar implementações que melho-

rem a eficiência e reduzam a área ocupada. Sendo o bloco de descodificação FEC o elemento que introduz mais latência no sistema, seria interessante utilizar outras arquitecturas conhecidas ou até desenvolver um algoritmo melhorado. As várias arquitecturas implementadas poderiam ser comparadas de modo a escolher a melhor.

- **Adaptação a ritmos de transmissão mais elevados** - Tirando partido do facto das tramas OTU1, OTU2 e OTU3 terem todas a mesma disposição de bytes, a adaptação dos blocos de recepção e transmissão para ritmos mais elevados passa essencialmente por alterar a dimensão do barramento de dados e possivelmente fazer algumas optimizações. O principal desafio para ritmos da ordem dos $10Gb/s$ ou $40Gb/s$ estará na interface física entre o domínio óptico e eléctrico.

- **Desmultiplexagem da carga-paga** - Os blocos de transmissão e recepção mencionados têm como função principal fazer a gestão dos cabeçalhos e correcção de erros da trama. O próximo passo será desmultiplexar os sinais transportados na carga-paga. Para aceder à carga-paga é necessário saber o tipo de dados que estão a ser transportados.

- **Implementação do sistema num ASIC** - Os FPGAs são dispositivos ideais para desenvolvimento e testes de circuitos digitais, no entanto têm um custo relativamente elevado. Com vista a comercialização do equipamento, uma solução mais económica é criar circuitos dedicados (ASIC). Apesar da grande complexidade do sistema, a substituição dos dispendiosos FPGAs por ASICs seria uma hipótese a explorar.

Bibliografia

[AGI00] AGILENT. *Agilent Point the Way to SDH*. 2000. `http://cp.literature.agilent.com/litweb/pdf/5980-2503EN.pdf` (acedido em Maio de 2009).

[AGI01] AGILENT. *Agilent Makes Light Work in the OTN*. 2001. `http://cp.literature.agilent.com/litweb/pdf/5988-4004EN.pdf` (acedido em Maio de 2009).

[BC02] N. Benvenuto and G. Cherubini. *Algorithms for Communications Systems and Their Applications*. wiley, 4th edition, 2002.

[BDM81] S.A. Butman, L.J. Deutsch, and R.L. Miller. *Performance of Concatenated Codes for Deep Space Missions*. TDA progress report 42-63, 1981.

[BGT93] C. Berrou, A. Glavieux, and P. Thitimajshima. *Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes*. IEEE International Conference on Communications, 1993 Geneva, 2:1064–1070, 1993.

[Bla83] R.E. Blahut. *Theory and Practice of Error Control Codes*. Addison-Wesley, Reading (Mass), 1983.

[CCR02] A.B. Carlson, P.B. Crilly, and J.C. Rutledge. *Communication Systems - An introduction to signals and noise in electrical communication*. McGraw-Hill, Boston, 4th edition, 2002.

[Che06] S. Chen. *Future Development Trends of Optical Transport Network Infrastructure*. PhD thesis, University of Wollongong, 2006.

[Cla02] C.K.P. Clarke. *Reed-Solomon Error Correction - R&D white paper (WHP031)*. BBC, Julho 2002. `http://downloads.bbc.co.uk/rd/pubs/whp/whp-pdf-files/WHP031.pdf` (acedido em Maio de 2009).

[Ins02] New Wave Instruments. *Linear Feedback Shift Registers - LFSR Generator Implementations*. Website, 2002. `http://www.newwaveinstruments.com/resources/articles/m_sequence_linear_feedback_shift_register_lfsr.htm` (acedido em Maio de 2009).

[IT00] ITU-T. *G.975 - Forward Error Correction for Submarine Systems*. Outubro 2000. `http://www.itu.int/rec/T-REC-G.975/en` (acedido em Maio de 2009).

[IT01] ITU-T. *G.872 - Architecture of Optical Transport Networks*. November 2001. `http://www.itu.int/rec/T-REC-G.872/en` (acedido em Maio de 2009).

[IT02] ITU-T. *G.694.1 - Spectral Grids for WDM Applications: DWDM frequency grid*. Junho 2002. `http://www.itu.int/rec/T-REC-G.694.1/en` (acedido em Maio de 2009).

[IT03a] ITU-T. *G.694.2 - Spectral Grids for WDM Applications: CWDM wavelength grid*. Dezembro 2003. `http://www.itu.int/rec/T-REC-G.694.2/en` (acedido em Maio de 2009).

[IT03b] ITU-T. *G.709 - Interfaces for the Optical Transport Network (OTN)*. Março 2003. `http://www.itu.int/rec/T-REC-G.709/en` (acedido em Maio de 2009).

[IT06] ITU-T. *G.798 - Characteristics of Optical Transport Network Hierarchy Equipment Functional Blocks*. Dezembro 2006. `http://www.itu.int/rec/T-REC-G.798/en` (acedido em Maio de 2009).

[IT07] ITU-T. *G.707 - Network Node Interface for the Synchronous Digital Hierarchy (SDH)*. Janeiro 2007. `http://www.itu.int/rec/T-REC-G.707/en` (acedido em Maio de 2009).

[JDS08] JDSU. *OTN - EfFECtive Networks*. 2008. `http://www.jdsu.com/product-literature/otn-fec_po_opt_tm_ae.pdf` (acedido em Maio de 2009).

[Kaz06] K. Kazi. *Optical Networking Standards - A comprehensive guide for professionals*. Springer, New York, 2006.

[Kei00] G. Keiser. *Optical Fiber Communications*. McGraw-Hill, Boston, 3rd edition, 2000.

[Kim03] H.J. Kim. *M-file of LFSR sequence generator*. Website, 2003. `http://www.watermarkingworld.org/WMMLArchive/0302/msg00023.html` (acedido em Junho de 2009).

[Lee05] H. Lee. *A High-Speed Low-Complexity Reed-Solomon Decoder for Optical Communications*. IEEE Transactions on Circuits and Systems II: Express Briefs, 52(8):461–465, 2005.

[Lei04] M.J. Leitão. *Tecnologias e Sistemas de Comunicação - Rede de Transporte*. FEUP, Porto, 2004.

[Mar07] E. Martins. *Electrónica III Cap. 6 - Circuitos Sequenciais*. DETI/UA, Aveiro, 2007. `http://elearning.ua.pt` (acedido em Dezembro de 2007).

[MG02] E. Marconetti and R. Guenard. *A fully programmable Reed Solomon 8-bit codec based on a Re-shaped Berlekamp Massey algorithm*. 2002 IEEE International Symposium on Circuits and Systems, 5:553–556, 2002.

[Moo05] T.K. Moon. *Error Correction Coding Mathematical Methods and Algorithms*. Wiley, 2005.

[NOR08] NORTEL. *Solving the 100Gbps Transmission Challenge*. 2008. `http://www.nortel.com/solutions/wireless/collateral/nn123688.pdf` (acedido em Maio de 2009).

[PH05] D.A. Patterson and J.L. Hennessy. *Computer Organization and Design - The hardware/software interface*. Elsevier, Amsterdam, 3rd edition, 2005.

[Pir04] J. Pires. *Redes de Telecomunicações - Redes Ópticas*. IST, Lisboa, 2004.

[PM07] R.G. Paiva and S.S. Marczak. *Desenvolvimento de Módulos de Hardware para Recepção e Transmissão de Quadros OTN*. FACIN/PUCRS, Porto Alegre, Dezembro 2007.

[RL00] A. Raghupathy and K.J.R. Liu. *Algorithm-Based Low-Power High-Speed Reed-Solomon Decoder Design*. IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, 47(11):1254–1270, 2000.

[Roc07a] J.F. Rocha. *Sistemas de Comunicação II - Códigos Correctores de erros*. DETI/UA, Aveiro, 2007. `http://elearning.ua.pt` (acedido em Janeiro de 2008).

[Roc07b] J.F. Rocha. *Sistemas de Comunicação II - Multiplex por divisão no tempo*. DETI/UA, Aveiro, 2007. `http://elearning.ua.pt` (acedido em Janeiro de 2008).

[RS02] R. Ramaswami and K.N. Sivarajan. *Optical Networks - A practical perspective*. Morgan Kaufmann, San Francisco, 2nd edition, 2002.

[RST91] I.S. Reed, M.T. Shih, and T.K. Truong. *VLSI design of inverse-free Berlekamp-Massey algorithm*. IEE Proceedings-E Computers and Digital Techniques, 138:295–298, 1991.

[Saw02] N. Sawyer. *Application Note: Virtex and Virtex-II Families - SONET and OTN Scramblers/Descramblers*. Xilinx, Novembro 2002. `http://www.xilinx.com/xapp/xapp652.pdf` (acedido em Maio de 2009).

[Sch06] A. Schubert. *G.709 - The Optical Transport Network*. JDSU, 2006. `http://www.jdsu.com/product-literature/g709otn_wp_opt_tm_ae.pdf` (acedido em Junho de 2009).

[Sha48] C.E. Shannon. *A Mathematical Theory of Communication*. Bell Systems Technical Journal, 27:379–423, 1948.

[SR07] A.H. Silva and T.A. Rodolfo. *Implementação de uma Arquitetura Reed-Solomon para uso em Redes OTN 10.7 Gbps*. FACIN/PUCRS, Porto Alegre, Dezembro 2007.

[SS01] D.V. Sarwate and N.R. Shanbhag. *High-Speed Architectures for Reed-Solomon Decoders*. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 9(5):641–655, 2001.

[Wal05] T.P. Walker. *Optical Transport Network (OTN) Tutorial*. ITU-T, 2005. `http://www.itu.int/ITU-T/studygroups/com15/otn/OTNtutorial.pdf` (acedido em Junho de 2009).

[WB94] S.B. Wicker and V.K. Bhargava. *Reed-Solomon Codes and their Applications*. IEEE, New York, 1994.

[WY05] K.C.C. Wai and S.J. Yang. *Field Programmable Gate Array Implementation of Reed-Solomon Code, RS(255,239)*. New York Workshop, Outubro 2005.

[Xil07] Xilinx. *VIRTEX-5 FPGAs - The Ultimate System Integration Platform*. 2007. `http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex5/V5brochure.pdf` (acedido em Maio de 2009).

[Xil09a] Xilinx. *Virtex-5 Family Overview - DS100 (v5.0)*. Fevereiro 2009. `http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf` (acedido em Junho de 2009).

[Xil09b] Xilinx. *Virtex-5 FPGA RocketIO GTP Transceiver User Guide - UG196 (v2.0)*. Junho 2009. `http://www.xilinx.com/support/documentation/user_guides/ug196.pdf` (acedido em Junho de 2009).

[Xil09c] Xilinx. *Virtex-5 FPGA User Guide - UG190 (v4.7).* Maio 2009. `http://www.xilinx.com/support/documentation/user_guides/ug190.pdf` (acedido em Junho de 2009).

[Xil09d] Xilinx. *XST User Guide - UG627 (v11.1).* Abril 2009. `http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/xst.pdf` (acedido em Junho de 2009).